(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 036 390 B1

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
03.11.2004 Bulletin 2004/45

(21) Application number: **98957726.7**

(22) Date of filing: **10.11.1998**

(51) Int Cl.[7]: **G09G 5/08**, G06K 11/18, G06F 3/00, G06F 3/033

(86) International application number:
PCT/US1998/023852

(87) International publication number:
WO 1999/026230 (27.05.1999 Gazette 1999/21)

(54) **Method of controlling of a force feedback device in a multi-tasking graphical host environment**

Methode der Steuerung einer Kraft-Rückkopplung-Vorrichtung in einer multi-tasking grafischen Wirtsumgebung

Méthode de commande d'un dispositif à retour de force dans un environnement d'hôte graphique multitaches

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR IE IT LI LU MC NL PT SE

(30) Priority: **14.11.1997 US 970953**

(43) Date of publication of application:
20.09.2000 Bulletin 2000/38

(73) Proprietor: **Immersion Corporation**
San Jose, CA 95131 (US)

(72) Inventors:
- **BRAUN, Adam, C.**
Sunnyvale, CA 94087 (US)
- **BEAMER, Jonathan, L.**
Menlo Park, CA 94025 (US)
- **ROSENBERG, Louis, B.**
Pleasanton, CA 94566 (US)
- **CHANG, Dean, C.**
Mountain View, CA 94043 (US)

(74) Representative: **Fiener, Josef et al**
Patentanw. J. Fiener et col.,
P.O. Box 12 49
87712 Mindelheim (DE)

(56) References cited:
WO-A-97/21160
US-A- 4 868 549
US-A- 5 530 455
US-A- 5 550 562

EP 1 036 390 B1

Description

BACKGROUND OF THE INVENTION

[0001] The present invention relates generally to interface devices for allowing humans to interface with computer systems, and more particularly to computer interface devices that allow the user to provide input to computer systems and provide force feedback to the user.

[0002] Computer systems are used extensively to implement many applications, such as word processing, data management, simulations, games, and other tasks. A computer system typically displays a visual environment to a user on a display screen or other visual output device. Users can interact with the displayed environment to perform functions on the computer, play a game, experience a simulated environment, use a computer aided design (CAD) system, etc. One visual environment that is particularly common is a graphical user interface (GUI). GUI's present visual images which describe various graphical metaphors of a program or operating system implemented on the computer. Common operating systems using GUI's include the Windows® operating system from Microsoft Corporation and the MacOS operating system from Apple Computer, Inc. The user typically moves a displayed, user-controlled graphical object, such as a cursor or pointer, across a computer screen and onto other displayed graphical objects or predefined screen regions, and then inputs a command to execute a given selection or operation. The objects or regions ("targets") can include, for example, icons, windows, pull-down menus, buttons, and scroll bars. Most GUIís are currently 2-dimensional as displayed on a computer screen; however, three-dimensional (3-D) GUI's that present simulated 3-D environments on a 2-D screen can also be provided. Other programs or environments that may provide user-controlled graphical objects such as a cursor or a "view" controlled by the user include graphical "web pages" or other environments offered on the World Wide Web of the Internet, CAD programs, video games, virtual reality simulations, etc.

[0003] The user interaction with and manipulation of the computer environment is achieved using any of a variety of types of human-computer interface devices that are connected to the computer system controlling the displayed environment. Such a method is disclosed in WO 97/21160. In most systems, the computer updates the environment in response to the user's manipulation of a user-manipulatable physical object ("user object") that is included in the interface device, such as a mouse, joystick, etc. The computer provides feedback to the user utilizing the display screen. A computer mouse is a common user object used to interact with a GUI or other graphical environment. A mouse (and other mouse-type devices such as a track ball) is typically used as a position control device in which displacement of the mouse in a planar workspace (e.g. on a mouse pad) is directly correlated to displacement of the user-controlled graphical object, such as a cursor, displayed on the screen.

[0004] Force feedback interface devices allow a user to experience forces on the manipulated user object based on interactions and events within the displayed graphical environment. Typically, computer-controlled actuators are used to output forces on the user object in provided degrees of freedom to simulate various sensations, such as an obstruction force when moving the cursor into a wall, a damping force to resist motion of the cursor, and a spring force to bias the cursor to move back toward a starting position of the spring. Force feedback devices can be implemented in many forms, such as a joystick, mouse, steering wheel, etc.

[0005] When implementing force feedback sensations in a GUI of an operating system, several problems can arise. One problem is the use of force feedback when multiple application programs are simultaneously running in a multi-tasking environment on the host computer. Most operating systems allow such multi-tasking, for example, to allow a user to interact with one application while one or more applications are also running, receiving data, outputting data, or performing other tasks. For example, in the Windows™ operating system, one application is the "active" application that typically displays an active window in the GUI. The user can manipulate the functions of the active application using the cursor. Other inactive applications are also running and may have inactive windows displayed in the GUI. The user can switch to a different application by clicking the cursor in an inactive window, for example, which causes the new application to be the active application and the formerly active application to become inactive.

[0006] Each application run by an operating system may have its own set of force sensations that it needs to command to the force feedback device. Thus, one application may need to command spring, force, vibration, and texture force sensations, while a different application may need to command spring, damper, and jolt force sensations. The force feedback device typically cannot store all possible force sensations for each application running in the operating system, so there is a problem of which force sensations the force feedback device should store and implement at any one time. In addition, if two of the multi-tasking applications command conflicting force sensations, the force feedback device needs to choose one of the force sensations to output, and there currently is no system or method of doing so.

[0007] A different problem occurs when using a force feedback device with a GUI. Traditional mouse controllers used with GUI's are relative position reporting devices, i.e., they report only changes in position of the mouse to the host computer, which the host computer uses to calculate a new position for the cursor on the screen. Many force feedback devices, in contrast, are typically absolute position reporting devices which report an absolute position of the cursor,

such as screen coordinates, to the host computer. This is because the force feedback device needs to know the cursor position to accurately determine when forces are to be applied and to accurately calculate the forces. However, it would be desirable in some instances to have a relative position reporting force feedback device, since the host computer is standardized to receive and interpret relative positions at the most basic level. Furthermore, such a relative device would permit the host computer to perform needed adjustments to cursor position, such as ballistics calculations which modify cursor position based on mouse velocity to provide enhanced control. If the host computer performs such adjustments, the force feedback device processors are relieved of computational burden. In addition, some types of interface devices such as trackballs are better suited to relative position reporting since an absolute, limited workspace is not easily defined for these devices.

[0008] Another problem occurs when force feedback is implemented with a GUI or other graphical environment and the graphical environment changes resolution or aspect ratio. For example, if a resolution of 640 x 480 is being displayed by the host computer on a screen, the force feedback device assumes that graphical objects in the GUI have a size proportional to screen dimensions and outputs forces accordingly. However, when the resolution is changed, the objects displayed on the screen change size in proportion to the screen dimensions. The force feedback device continues to check for conditions and generate forces as if the old resolution were active, resulting in forces that do not correlate with displayed interactions on the screen. The aspect ratio of a display screen can also change, e.g., when two screens are used to provide double the amount of displayed area in the GUI, the aspect ratio doubles in one dimension. Using prior art force feedback devices, forces can become distorted from such an aspect ratio change. For example, a circle object displayed on the screen may have forces at its borders that feel like an ellipse to the user of the interface device, since the aspect ratios of the screen and the mouse workspace are different.

SUMMARY OF THE INVENTION

[0009] The present invention is directed to a method which allows control of a force feedback device in a multi-tasking graphical host environment.

[0010] This object is achieved by a method according to claim 1.

[0011] More specifically, a method of the present invention interfaces a multi-tasking graphical environment implemented on a host computer with a force feedback interface device coupled to the host computer, where multiple application programs may run in the multi-tasking environment. A context is created for association with each application program running in the multi-tasking graphical environment. Force effect commands are received from the application programs, where the force effect commands command the force feedback interface device to output a force effect specified by the command. The force effect commands are stored into the contexts. Each context is associated with one of the application programs running on the host computer, and each force effect command is stored in a context associated with the application program that sent the force effect command. The force effect commands in the context of a particular application program are sent to the force feedback device when that particular application program is active in the multi-tasking environment. Preferably, the force effect commands in contexts of inactive application programs are not sent to the force feedback device. Thus, only the active application program may command forces on the force feedback device.

[0012] When the application program becomes inactive and a new application program becomes active, new force effect commands are sent to the force feedback device to replace the force effect commands of the formerly active application. Preferably, a background application is provided which also provides force effects to the force feedback device and may output forces on the device even when not active. Events are also provided which allow a graphical action such as an interaction of the cursor in the graphical environment with another object to cause an event notification to be sent to the application program. Preferably, only the active and background application programs may receive events.

[0013] Also described herein are several force feedback sensations and structures, including enclosures, textures, and grids. For example, an enclosure is a rectangular or elliptical shape having walls that provide forces to an associated graphical object such as an icon or window. The enclosure includes walls, where each wall may be associated with a force. In one embodiment, a particular type of enclosure is provided to modify the forces of a different enclosure to prevent conflicts between the forces of overlapping or closely-positioned enclosures.

[0014] The present invention provides several embodiments. The architecture on the host computer allows multi-tasking application programs to interface with the force feedback device without conflicts. The force feedback device provides both relative position reporting and absolute position reporting to allow great flexibility. A relative position reporting device allows maximum compatibility with existing software. Information such as ballistic parameters and screen size sent from the host to the force feedback device allow accurate mouse positions and cursor positions to be determined in the force feedback environment.

[0015] These and other advantages of the present invention will become apparent to those skilled in the art upon a reading of the following specification of the invention and a study of the several figures of the drawing.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016]

Figure 1 is a perspective view of one embodiment of a mouse interface system suitable for use with the present invention;

Figure 2 is a perspective view of an embodiment of a mechanism suitable for the interface system of Figure 1;

Figure 3 is a block diagram of the system of Figure 1 for controlling a force feedback interface device;

Figure 4 is a block diagram of an architecture for a host computer providing multiple application programs communicating with the force feedback device;

Figure 5 is a diagrammatic illustration of a background application program control panel allowing a user to characterize background forces;

Figures 6a, 6b and 6c are diagrammatic illustrations of embodiments of an enclosure force effect;

Figure 7 is a diagrammatic illustration of a texture force effect;

Figure 8 is a diagrammatic illustration of a grid force effect;

Figure 9 is a block diagram illustrating a preferred embodiment of implementing a force feedback device and system;

Figure 10 is a flow diagram illustrating a method of implementing position reporting for the embodiment of Figure 9;

Figure 11 is a flow diagram illustrating a method of determining cursor position and indexing in the embodiment of Figures 9 and 10;

Figure 12 is a block diagram illustrating a second embodiment of implementing a force feedback device and system;

Figure 13 is a flow diagram illustrating a method of implementing position reporting for the embodiment of Figure 12; and

Figures 14 and 15 are diagrammatic illustrations showing the error correction between cursor positions from host computer and force feedback device in the embodiment of Figure 13.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

[0017] FIGURE 1 is a perspective view of a force feedback mouse interface system 10 of the present invention, capable of providing input from the user to a host computer based on the useris manipulation of the mouse and capable of providing force feedback to the user of the mouse system based on events occurring in an environment implemented by the host computer. Mouse system 10 includes an interface device 11 that includes a user manipulatable object or manipulandum 12 and an interface 14, and a host computer 18.

[0018] User object (or "manipulandum") 12 is an physical object that is preferably grasped or gripped and manipulated by a user. By ìgrasp,î it is meant that users may releasably engage a portion of the object in some fashion, such as by hand, with their fingertips, etc. For example, images are displayed and/or modified on a display screen 20 of the computer system 18 in response to such manipulations. In the described embodiment, user object 12 is a mouse 12 that is shaped so that a userís fingers or hand may comfortably grasp the object and move it in the provided degrees of freedom in physical space. For example, a user can move mouse 12 to correspondingly move a computer generated graphical object, such as a cursor or other image, in a graphical environment provided by computer 18. The available degrees of freedom in which mouse 12 can be moved are determined from the interface 14, described below. In addition, mouse 12 preferably includes one or more buttons 15 to allow the user to provide additional commands to the computer system.

[0019] It will be appreciated that a great number of other types of user manipulable objects can be used with the method and apparatus of the present invention in place of or in addition to mouse 12. For example, such objects may

include a sphere, a puck, a joystick, cubical- or other-shaped hand grips, a fingertip receptacle for receiving a finger or a stylus, a flat planar surface like a plastic card having a rubberized, contoured, and/or bumpy surface, a handheld remote control used for controlling web pages or other devices, or other objects. In one embodiment, a small fingertip joystick can be provided, where a small stick is moved in a small planar region with a userís fingertips. Spring forces can be provided by the actuators of the device 11 to bias the stick (or any type of joystick) toward the center of the planar region to simulate a spring return on the joystick. Since fingertips are used, output forces need not be as high a magnitude as in other embodiments. Also, mouse 12 can be provided with such a centering spring bias, e.g. when used like a joystick in gaming applications.

[0020] Interface 14 interfaces mechanical and electrical input and output between the mouse 12 and host computer 18 implementing the application program, such as a GUI, simulation or game environment. Interface 14 provides multiple degrees of freedom to mouse 12; in the preferred embodiment, two linear, planar degrees of freedom are provided to the mouse, as shown by arrows 22. In other embodiments, greater or fewer degrees of freedom can be provided, as well as rotary degrees of freedom. For many applications, mouse 12 need only be moved in a very small workspace area.

[0021] In a preferred embodiment, the user manipulates mouse 12 in a planar workspace, much like a traditional mouse, and the position of mouse 12 is translated into a form suitable for interpretation by position sensors of the interface 14. The sensors track the movement of the mouse 12 in planar space and provide suitable electronic signals to an electronic portion of interface 14. The interface 14 provides position information to host computer 18. In addition, host computer 18 and/or interface 14 provide force feedback signals to actuators coupled to interface 14, and the actuators generate forces on members of the mechanical portion of the interface 14 to provide forces on mouse 12 in provided or desired degrees of freedom. The user experiences the forces generated on the mouse 12 as realistic simulations of force sensations such as jolts, springs, textures, enclosures, circles, ellipses, grids, vibrations, ìbarrierî forces, and the like.

[0022] The electronic portion 26 of interface 14 may couple the mechanical portion 24 of the interface to the host computer 18. The electronic portion 26 is preferably included within the housing 21 of the interface 14 or, alternatively, the electronic portion may be included in host computer 18 or as a separate unit with its own housing. More particularly, interface 14 includes a local microprocessor distinct and separate from any microprocessors in the host computer 18 to control force feedback on mouse 12 independently of the host computer, as well as sensor and actuator interfaces that convert electrical signals to appropriate forms usable by the mechanical portion of interface 14 and host computer 18.

[0023] For example, a rigid surface is generated on computer screen 20 and a computer object (e.g., cursor) controlled by the user collides with the surface. In a preferred embodiment, high-level host commands can be used to provide the various forces associated with the rigid surface. The local control mode using a local microprocessor in interface 14 can be helpful in increasing the response time for forces applied to the user object, which is essential in creating realistic and accurate force feedback. For example, it is preferable that host computer 18 send a ispatial representationî to the local microprocessor, which is data describing the locations of some or all the graphical objects displayed in a GUI or other graphical environment which are associated with forces and the types/characteristics of these graphical objects. The microprocessor can store such a spatial representation in local memory, and thus will be able to determine interactions between the user object and graphical objects (such as the rigid surface) independently of the host computer. In addition, the microprocessor can be provided with the necessary instructions or data to check sensor readings, determine cursor and target positions, and determine output forces independently of host computer 18. The host could implement program functions (such as displaying images) when appropriate, and synchronization commands can be communicated between the microprocessor and host 18 to correlate the microprocessor and host processes. Such commands and related functionality is discussed in greater detail below. Alternatively, the computer 18 can directly send force feedback signals to the interface 14 to generate forces on mouse 12. A suitable embodiment of the electrical portion of interface 14 is described in detail with reference to Figure 3.

[0024] The interface 14 can be coupled to the computer 18 by a bus 17, which communicates signals between interface 14 and computer 18 and also, in the preferred embodiment, provides power to the interface 14 (e.g. when bus 17 includes a USB interface). In other embodiments, signals can be sent between interface 14 and computer 18 by wireless transmission/reception. In preferred embodiments of the present invention, the interface 14 serves as an input/output (I/O) device for the computer 18. The interface 14 can also receive inputs from other input devices or controls that are associated with mouse system 10 and can relay those inputs to computer 18. For example, commands sent by the user activating a button on mouse 12 can be relayed to computer 18 by interface 14 to implement a command or cause the computer 18 to output a command to the interface 14.

[0025] Host computer 18 is preferably a personal computer or workstation, such as an IBM-PC compatible computer or Macintosh personal computer, or a SUN or Silicon Graphics workstation. For example, the computer 18 can operate under the Windowsô or MS-DOS operating system in conformance with an IBM PC AT standard. Alternatively, host computer system 18 can be one of a variety of home video game systems commonly connected to a television set,

such as systems available from Nintendo, Sega, or Sony. In other embodiments, host computer system 18 can be a ìset top boxi which can be used, for example, to provide interactive television functions to users, or a ìnetwork-î or ìinternet-computerî which allows users to interact with a local or global network using standard connections and protocols such as used for the Internet and World Wide Web. Host computer preferably includes a host microprocessor, random access memory (RAM), read only memory (ROM), input/output (I/O) circuitry, and other components of computers well-known to those skilled in the art.

[0026] Host computer 18 preferably implements one or more application programs ("applications") with which a user is interacting via mouse 12 and other peripherals, if appropriate, and which can include force feedback functionality. For example, the host application programs can include a simulation, video game, Web page or browser that implements HTML or VRML instructions, word processor, drawing program, spreadsheet, scientific analysis program, or other application program that utilizes input of mouse 12 and outputs force feedback commands to the mouse 12. Typically, an operating systems such as Windowsô, MS-DOS, MacOS, Unix, is also running on the host computer and preferably includes its own force feedback functionality. In one preferred embodiment, the operating system and application programs utilize a graphical user interface (GUI) to present options to a user, display data and images, and receive input from the user. In the preferred embodiment, multiple applications can run simultaneously in a multitasking environment of the host computer, as is detailed below. Herein, computer 18 may be referred as displaying igraphical objectsi or ìcomputer objects.î These objects are not physical objects, but are logical software unit collections of data and/or procedures that may be displayed as images by computer 18 on display screen 20, as is well known to those skilled in the art. A displayed cursor or a simulated cockpit of an aircraft might be considered a graphical object. The host application program checks for input signals received from the electronics and sensors of interface 14, and outputs force values and/or commands to be converted into forces on mouse 12. Suitable software drivers which interface such simulation software with computer input/output (I/O) devices are available from Immersion Human Interface Corporation of San Jose, California.

[0027] Display device 20 can be included in host computer 18 and can be a standard display screen (LCD, CRT, etc.), 3-D goggles, or any other visual output device. Typically, the host application provides images to be displayed on display device 20 and/or other feedback, such as auditory signals. For example, display screen 20 can display images from a GUI. Images describing a moving, first person point of view can be displayed, as in a virtual reality game. Or, images describing a third-person perspective of objects, backgrounds, etc. can be displayed. Alternatively, images from a simulation, such as a medical simulation, can be displayed, e.g., images of tissue and a representation of a manipulated user object 12 moving through the tissue, etc.

[0028] There are two primary icontrol paradigmsî of operation for mouse system 10: position control and rate control. Position control is the more typical control paradigm for mouse and similar controllers, and refers to a mapping of mouse 12 in which displacement of the mouse in physical space directly dictates displacement of a graphical object. Under a position control mapping, the computer object does not move unless the user object is in motion. Also, iballisticsî or other non-linear adjustments to cursor position can be used in which, for example, slow motions of the mouse have a different scaling factor for cursor movement than fast motions of the mouse, to allow more control of short cursor movements.

[0029] As shown in Figure 1, the host computer may have its own iscreen frameî 28 (or host frame) which is displayed on the display screen 20. In contrast, the mouse 12 has its own ìdevice frameî (or local frame) 30 in which the mouse 12 is moved. In a position control paradigm, the position (or change in position) of a user-controlled graphical object, such as a cursor, in host frame 30 corresponds to a position (or change in position) of the mouse 12 in the local frame 28.

[0030] Rate control is also used as a control paradigm. This refers to a mapping in which the displacement of the mouse 12 is abstractly mapped to motion of a computer object under control. There is not a direct physical mapping between physical object (mouse) motion and computer object motion. Thus, most rate control paradigms are fundamentally different from position control in that the user object can be held steady at a given position but the controlled computer object is in motion at a commanded or given velocity, while the position control paradigm only allows the controlled computer object to be in motion if the user object is in motion.

[0031] The mouse interface system 10 is useful for both position control (ìisotonicî) tasks and rate control (ìisometricî) tasks. For example, as a traditional mouse, the position of mouse 12 in its local frame 30 workspace can be directly mapped to a position of a cursor in host frame 28 on display screen 20 in a position control paradigm. Alternatively, the displacement of mouse 12 in a particular direction against an opposing output force can command rate control tasks in an isometric mode.

[0032] Mouse 12 is preferably supported and suspended above a grounded pad 32 by the mechanical portion of interface 14, described below. Pad 32 or similar surface is supported by grounded surface 34. Pad 32 (or alternatively grounded surface 34) provides additional support for the mouse and relieve stress on the mechanical portion of interface 14. In addition, a wheel, roller, Teflon pad or other device can be used to support the mouse.

[0033] Mouse 12 can be used, for example, to control a computer-generated graphical object such as a cursor displayed in a graphical computer environment, such as a GUI. The user can move the mouse in 2D planar workspace

to move the cursor to graphical objects in the GUI or perform other tasks. In other graphical environments, such as a virtual reality video game, a user can be controlling a computer player or vehicle in the virtual environment by manipulating the mouse 12. The computer system tracks the position of the mouse with sensors as the user moves it. The computer system may also provide force feedback commands to the mouse, for example, when the user moves the graphical object against a generated surface such as an edge of a window, a virtual wall, etc. It thus appears and feels to the user that the mouse and the graphical object are contacting real surfaces.

[0034] The mouse system 10 may also include an indexing function or iindexing modeî which allows the user to redefine the offset between the positions of the mouse 12 in the local frame and a user-controlled graphical object, such as a cursor, in the host frame displayed by host computer 18. In one implementation, the user may reposition the mouse 12 without providing any input to the host computer, thus allowing the user to redefine the offset between the objectís position and the cursorís position. Such indexing is achieved through an input device such as a button, switches, sensors, or other input devices. As long as the indexing device is activated, the mouse 12 is in indexing mode; when the button is released (or indexing mode otherwise exited), the position of the cursor is again controlled by the mouse 12. A hand weight switch can also be provided which inherently causes indexing when the user removes hand or finger weight from mouse 12.

[0035] A different way to allow indexing is to provide a combined position control and rate control device which allows different forms of control of the cursor depending on the position of the mouse in its workspace.

[0036] FIGURE 2 is a perspective view of a preferred embodiment of the mouse device 11 with the cover portion of housing 21 and the grounded pad 32 removed. Mechanical linkage 40 provides support for mouse 12 and couples the mouse to a grounded surface 34, such as a tabletop or other support. Linkage 40 is, in the described embodiment, a 5-member (or ì5-barî) linkage. Fewer or greater numbers of members in the linkage can be provided in alternate embodiments.

[0037] Ground member 42 of the linkage 40 is a base for the support of the linkage and is coupled to or resting on a ground surface 34. The ground member 42 in Figure 2 is shown as a plate or base that extends under mouse 12. The members of linkage 40 are rotatably coupled to one another through the use of rotatable pivots or bearing assemblies, all referred to as ìbearingsî herein. Base member 44 is rotatably coupled to ground member 42 by a grounded bearing 52 and can rotate about an axis A. Link member 46 is rotatably coupled to base member 44 by bearing 54 and can rotate about a floating axis B, and base member 48 is rotatably coupled to ground member 42 by bearing 52 and can rotate about axis A. Link member 50 is rotatably coupled to base member 48 by bearing 56 and can rotate about floating axis C, and link member 50 is also rotatably coupled to link member 46 by bearing 58 such that link member 50 and link member 46 may rotate relative to each other about floating axis D.

[0038] Linkage 40 is formed as a five-member closed-loop chain. Each member in the chain is rotatably coupled to two other members of the chain to provide mouse 12 with two degrees of freedom, i.e., mouse 12 can be moved within a planar x-y workspace. Mouse 12 is coupled to link members 46 and 50 by rotary bearing 58. and may rotate at least partially about axis D. A pad or other support can be provided under mouse 12 to help support the mouse.

[0039] Transducer system 41 is used to sense the position of mouse 12 in its workspace and to generate forces on the mouse 12. Transducer system 41 preferably includes sensors 62 and actuators 64. The sensors 62 collectively sense the movement of the mouse 12 in the provided degrees of freedom and send appropriate signals to the electronic portion of interface 14. Sensor 62a senses movement of link member 48 about axis A, and sensor 62b senses movement of base member 44 about axis A. These sensed positions about axis A allow the determination of the position of mouse 12 using known constants such as the lengths of the members of linkage 40 and using well-known coordinate transformations.

[0040] Sensors 62 are, in the described embodiment, grounded optical encoders that sense the intermittent blockage of an emitted beam. A grounded emitter/detector portion 71 includes an emitter that emits a beam which is detected by a grounded detector. A moving encoder disk portion or ìarcî 74 is provided at the end of members 44 and 48 which each block the beam for the respective sensor in predetermined spatial increments and allows a processor to determine the position of the arc 74 and thus the members 44 and 48 by counting the spatial increments. Also, a velocity of members 44 and 48 based on the speed of passing encoder marks can also be determined.

[0041] Transducer system 41 also preferably includes actuators 64 to transmit forces to mouse 12 in space, i.e., in two (or more) degrees of freedom of the user object. The bottom housing plate 65 of actuator 64a is rigidly coupled to ground member 42 (or grounded surface 34) and a moving portion of actuator 64a (preferably a coil) is integrated into the base member 44. The actuator 64a transmits rotational forces to base member 44 about axis A. The housing 65 of the grounded portion of actuator 64b is rigidly coupled to ground member 42 or ground surface 34 through the grounded housing of actuator 64b, and a moving portion (preferably a coil) of actuator 64b is integrated into base member 48. Actuator 64b transmits rotational forces to link member 48 about axis A. The combination of these rotational forces about axis A allows forces to be transmitted to mouse 12 in all directions in the planar workspace provided by linkage 40 through the rotational interaction of the members of linkage 40. The integration of the coils into the base members 44 and 48 is advantageous to the present invention and is discussed below.

[0042] FIGURE 3 is a block diagram illustrating the electronic portion of interface 14 and host computer 18 suitable for use with the present invention. Mouse interface system 10 includes a host computer 18, electronic interface 26, mechanical portion 24, and mouse or other user object 12. Electronic interface 26, mechanical portion 24, and mouse 12 can also collectively be considered the " force feedback interface device" 11 that is coupled to the host computer.

[0043] As explained with reference to Figure 1, computer 18 is preferably a personal computer, workstation, video game console, or other computing or display device. Host computer system 18 commonly includes a host microprocessor 108, random access memory (RAM) 110, read-only memory (ROM) 112, input/output (I/O) electronics 114, a clock 116, a display device 20, and an audio output device 118. Host microprocessor 108 can include a variety of available microprocessors from Intel, AMD, Motorola, or other manufacturers. Microprocessor 108 can be single microprocessor chip, or can include multiple primary and/or co-processors. Microprocessor 108 preferably retrieves and stores instructions and other necessary data from RAM 110 and ROM 112 as is well known to those skilled in the art. In the described embodiment, host computer system 18 can receive sensor data or a sensor signal via a bus 120 from sensors of system 10 and other information. Microprocessor 108 can receive data from bus 120 using I/O electronics 114, and can use I/O electronics to control other peripheral devices. Host computer system 18 can also output commands to interface device 104 via bus 120 to cause force feedback for the interface system 10.

[0044] Clock 116 is a standard clock crystal or equivalent component used by host computer 18 to provide timing to electrical signals used by host microprocessor 108 and other components of the computer system 18. Display device 20 is described with reference to Figure 1. Audio output device 118, such as speakers, can be coupled to host microprocessor 108 via amplifiers, filters, and other circuitry well known to those skilled in the art. Host processor 108 outputs signals to speakers 118 to provide sound output to the user when an "audio event" occurs during the implementation of the host application program. Other types of peripherals can also be coupled to host processor 108, such as storage devices (hard disk drive, CD ROM drive, floppy disk drive, etc.), printers, and other input and output devices.

[0045] Electronic interface 26 is coupled to host computer system 18 by a bi-directional bus 120. The bi-directional bus sends signals in either direction between host computer system 18 and the interface device 104. Bus 120 can be a serial interface bus providing data according to a serial communication protocol, a parallel bus using a parallel protocol, or other types of buses. An interface port of host computer system 18, such as an RS232 serial interface port, connects bus 120 to host computer system 18. In another embodiment, an additional bus can be included to communicate between host computer system 18 and interface device 11. One preferred serial interface bus used in the present invention is the Universal Serial Bus (USB). The USB standard provides a relatively high speed serial interface that can provide force feedback signals in the present invention with a high degree of realism. USB can also source power to drive actuators 64 and other devices of the present invention. In addition, the USB standard includes timing data that is encoded along with differential data. Alternatively, Firewire (also called 1392) can be used as bus 120; or, the bus can be between an interface card in the host computer 18, where the interface card holds components of device 11 such as microprocessor 130.

[0046] Electronic interface 26 includes a local microprocessor 130, local clock 132, local memory 134, sensor interface 136, and actuator interface 138. Interface 26 may also include additional electronic components for communicating via standard protocols on bus 120. In various embodiments, electronic interface 26 can be included in mechanical portion 24, in host computer 18, or in its own separate housing. Different components of interface 26 can be included in portion 24 or host computer 18 if desired.

[0047] Local microprocessor 130 preferably coupled to bus 120 and may be closely linked to mechanical portion 24 to allow quick communication with other components of the interface device. Processor 130 is considered "local" to interface device 11, where "local" herein refers to processor 130 being a separate microprocessor from any processors 108 in host computer 18. "Local" also preferably refers to processor 130 being dedicated to force feedback and sensor I/O of the interface system 10, and being closely coupled to sensors and actuators of the mechanical portion 24, such as within the housing of or in a housing coupled closely to portion 24. Microprocessor 130 can be provided with software instructions to wait for commands or requests from computer host 18, parse/decode the command or request, and handle/control input and output signals according to the command or request. In addition, processor 130 preferably operates independently of host computer 18 by reading sensor signals and calculating appropriate forces from those sensor signals, time signals, and force processes selected in accordance with a host command, and output appropriate control signals to the actuators. A suitable microprocessor for use as local microprocessor 130 includes the 8X930AX by Intel; or alternatively the MC68HC711E9 by Motorola or the PIC16C74 by Microchip, for example. Microprocessor 130 can include one microprocessor chip, or multiple processors and/or coprocessor chips. In other embodiments, microprocessor 130 can include digital signal processor (DSP) functionality.

[0048] For example, in one host-controlled embodiment that utilizes microprocessor 130, host computer 18 can provide low-level force commands over bus 120, which microprocessor 130 directly transmits to the actuators. In a different local control embodiment, host computer system 18 provides high level supervisory commands to microprocessor 130 over bus 120, and microprocessor 130 manages low level force control loops to sensors and actuators in accordance with the high level commands and independently of the host computer 18. In the local control embodiment,

the microprocessor 130 can process inputted sensor signals to determine appropriate output actuator signals by following the instructions of a "force process" that may be stored in local memory and includes calculation instructions, formulas, force magnitudes, or other data. The force process can command distinct force sensations, such as vibrations, textures, jolts, or even simulated interactions between displayed objects. An "enclosure" host command can also be provided, which causes the microprocessor to define a box-like enclosure in a graphical environment, where the enclosure has sides characterized by wall and texture forces. The host can send the local processor a spatial layout of objects in the graphical environment so that the microprocessor has a mapping of locations of graphical objects like enclosures and can determine interactions with the cursor locally.

[0049] Sensor signals used by microprocessor 130 are also reported to host computer system 18, which updates a host application program and outputs force control signals as appropriate. For example, if the user moves mouse 12, the computer system 18 receives position and/or other signals indicating this movement and can move a displayed cursor in response. In an alternate embodiment, no local microprocessor 130 is included in interface system 10, and host computer 18 directly controls and processes all signals to and from the interface 26 and mechanical portion 24.

[0050] A local clock 132 can be coupled to the microprocessor 130 to provide timing data, similar to system clock 116 of host computer 18; the timing data might be required, for example, to compute forces output by actuators 64 (e.g., forces dependent on calculated velocities or other time dependent factors). In alternate embodiments using the USB communication interface, timing data for microprocessor 130 can be retrieved from the USB interface. Local memory 134, such as RAM and/or ROM, is preferably coupled to microprocessor 130 in interface 26 to store instructions for microprocessor 130 and store temporary and other data. Microprocessor 130 may also store calibration parameters in a local memory 134 such as an EEPROM. As described above, link or member lengths or manufacturing variations and/or variations in coil winding or magnet strength can be stored. Memory 134 may be used to store the state of the force feedback device, including a reference position, current control mode or configuration, etc.

[0051] Sensor interface 136 may optionally be included in electronic interface 26 to convert sensor signals to signals that can be interpreted by the microprocessor 130 and/or host computer system 18. For example, sensor interface 136 can receive signals from a digital sensor such as an encoder and convert the signals into a digital binary number representing the position of a member of mechanical apparatus 14. An analog to digital converter (ADC) in sensor interface 136 can convert a received analog signal to a digital signal for microprocessor 130 and/or host computer 18. Such circuits, or equivalent circuits, are well known to those skilled in the art. Alternately, microprocessor 130 or host 18 can perform these interface functions.

[0052] Actuator interface 138 can be optionally connected between the actuators 64 and microprocessor 130. Interface 138 converts signals from microprocessor 130 into signals appropriate to drive the actuators. Interface 138 can include power amplifiers, switches, digital to analog controllers (DACs), and other components. Such interfaces are well known to those skilled in the art. In alternate embodiments, interface 138 circuitry can be provided within microprocessor 130 or in the actuators.

[0053] In the described embodiment, power is supplied to the actuators 64 and any other components (as required) by the USB. Since the electromagnetic actuators of the described embodiment have a limited physical range and need only output, for example, about 3 ounces of force to create realistic force sensations on the user, very little power is needed. For example, one way to draw additional power from the USB is to configure device 11 to appear as more than one peripheral to host computer 18; for example, each provided degree of freedom of mouse 12 can be configured as a different peripheral and receive its own allocation of power. Alternatively, power from the USB can be stored and regulated by device 11 and thus used when needed to drive actuators 64. For example, power can be stored over time and then immediately dissipated to provide a jolt force to the user object 12. A battery or a capacitor circuit, for example, can store energy and discharge or dissipate the energy when power is required by the system and/or when enough power has been stored. Alternatively, a power supply 140 can optionally be coupled to actuator interface 138 and/or actuators 64 to provide electrical power. The power storage embodiment described above, using a battery or capacitor circuit, can also be used in non-USB embodiments to allow a smaller power supply 140 to be used.

[0054] Mechanical portion 24 is coupled to electronic portion 26 and preferably includes sensors 62, actuators 64, and linkage 40. These components are described in detail above. Sensors 62 sense the position, motion, and/or other characteristics of mouse 12 along one or more degrees of freedom and provide signals to microprocessor 130 including information representative of those characteristics. Typically, a sensor 62 is provided for each degree of freedom along which mouse 12 can be moved, or, a single compound sensor can be used for multiple degrees of freedom. Example of sensors suitable for embodiments described herein are rotary optical encoders, as described above, linear optical encoders, analog sensors such as potentiometers, or non-contact sensors, such as Hall effect magnetic sensors or an optical sensor such as a lateral effect photo diode having an emitter/detector pair. In addition, velocity sensors (e.g., tachometers) and/or acceleration sensors (e.g., accelerometers) for measuring acceleration can be used. Furthermore, either relative or absolute sensors can be employed.

[0055] Actuators 64 transmit forces to mouse 12 in one or more directions along one or more degrees of freedom in response to signals output by microprocessor 130 and/or host computer 18, i.e., they are " computer controlled." Typ-

ically, an actuator 64 is provided for each degree of freedom along which forces are desired to be transmitted. Actuators 64 can be the electromagnetic actuators described above, or can be other active actuators, such as linear current control motors, stepper motors, pneumatic/hydraulic active actuators, a torquer (motor with limited angular range); or passive actuators such as magnetic particle brakes, friction brakes, or pneumatic/hydraulic passive actuators, passive damper elements, an electrorheological fluid actuator, a magnetorheological fluid actuator. In addition, in voice coil embodiments, multiple wire coils can be provided, where some of the coils can be used to provide back EMF and damping forces. In some embodiments, all or some of sensors 62 and actuators 64 can be included together as a sensor/actuator pair transducer.

[0056] Mechanism 40 is preferably the five-member linkage 40 described above, but can also be one of several types of mechanisms. Mouse 12 can alternatively be a puck, joystick, or other device or article coupled to linkage 40, as described above.

[0057] Other input devices 141 can optionally be included in system 10 and send input signals to microprocessor 130 and/or host computer 18. Such input devices can include buttons, such as buttons 15 on mouse 12, used to supplement the input from the user to a GUI, game, simulation, etc. Also, dials, switches, voice recognition hardware (with software implemented by host 18), or other input mechanisms can be used.

[0058] Safety or "deadman" switch 150 is preferably included in interface device to provide a mechanism to allow a user to deactivate actuators 64 for safety reasons. In the preferred embodiment, the user must continually close safety switch 150 during manipulation of mouse 12 to activate the actuators 64. If, at any time, the safety switch is deactivated (opened) the actuators are deactivated while the safety switch is open. For example, one embodiment of safety switch is a capacitive contact sensor that detects when the user is contacting mouse 12. Alternatively, a mechanical switch, electrostatic contact switch, or optical switch located on mouse 12 or on a convenient surface of a housing 21 can be used. A z-axis force sensor, piezo electric sensors, force sensitive resistors, or strain gauges can also be used. The state of the safety switch can be sent to the microprocessor 130 or host 18. In one embodiment, mouse 12 includes a hand weight safety switch. The safety switch preferably deactivates any generated forces on the mouse when the mouse is not in use and/or when the user desires to deactivate output forces.

[0059] In some embodiments of interface system 10, multiple mechanical apparatuses 102 and/or electronic interfaces 100 can be coupled to a single host computer system 18 through bus 120 (or multiple buses 120) so that multiple users can simultaneously interface with the host application program (in a multi-player game or simulation, for example). In addition, multiple players can interact in the host application program with multiple interface systems 10 using networked host computers 18, as is well known to those skilled in the art. Also, the interface device 104 can be coupled to multiple host computers; for example, a local host computer can display images based on data received from a remote host computer coupled to the local host through a network.

Architecture of Host Computer

[0060] The host computer 18 interacts with interface device 11, in the present invention, using a number of different levels of controllers. These controllers are preferably implemented in software, e.g. program instructions or code, and such is the embodiment described herein; however, all or part of the controllers may also be implemented in hardware, where the conversion of functionality of the software to hardware is well known to those skilled in the art.

[0061] The architecture described herein is oriented towards providing force feedback functionality to a host computer connected to a force feedback interface device, where multiple applications may be running simultaneously on the host computer. Each application program may have its own set of force sensations to output to the device; since the device cannot implement all force sensations at any one time due to cost and hardware constraints, the forces commanded by each application must be organized by the architecture to take these limitations into account.

[0062] FIGURE 4 is a block diagram of a preferred architecture for the host computer to communicate with and control a force feedback interface device 11 with multiple application programs running on the host computer. Application programs 202 and 204 are concurrently running on the host computer 18. In the most typical implementation, one of the application programs is actively running in an operating system as the "active" application program that displays one or more active windows in the GUI (also known as the application program that is "in focus" or which has "keyboard focus"). The active window is typically the topmost displayed window in which input is provided by the user using the mouse-controlled cursor, a keyboard, or other peripheral. The other applications are "inactive" in that they are not receiving input from the user (although they may have a window displayed in the GUI which can be updated on the screen). The inactive applications may also receive input or send output. For example, the Windows™ operating system from Microsoft Corp. provides a multitasking or pseudo-multitasking environment in which programs run simultaneously; other operating systems such as Unix also offer multitasking. For example, a word processor may be the active application to receive input from the keyboard and display input characters in a displayed active window on display screen 20, while a inactive communication program may be receiving data over a network and saving the data on a storage device, or sending data out to be printed. Or, a inactive drawing program may be displaying a graphical

image in a window that is not currently active, while the user inputs text into the active word processor window. When the user moves the cursor over an inactive window and provides a command gesture such as clicking a button on a mouse, the inactive window becomes the active window and the former active window becomes inactive. The active application is also known as the "foreground" application, in the sense that its force sensations are being implemented by the force feedback device, as described below.

[0063] A master application 206 also is running on host computer 18 and is also known as the "background" force feedback application. This application is preferably a general purpose program that always runs inactively in the operating system and whose set of commanded forces are always available to be output and controlled on the interface device 11 and/or other devices. For example, a preferred embodiment of the master application is a "desktop" control panel for force feedback. An example of such a program is illustrated in FIGURE 5. A "mouse properties" dialog box 240 can be displayed which allows the user to specify force sensations 244 that are assigned to specified object types 242 in the displayed graphical environment, e.g. a graphical user interface (GUI). For example, the assigned force sensation 244 can be a vibration force having a specified frequency, magnitude, duration, etc.; a single "pop" or jolt at a specified time and duration; a "tick" that is a small jolt that is output based on movement of the user object/cursor or at regular intervals during an activity such as scrolling; a damping force condition that causes a resistance to mouse motion depending on the velocity of the user object 12 in its degrees of freedom or the cursor in the host frame on screen 20; or a spring that resists motion of the user object based on distance to an origin of the spring.

[0064] The force sensations specified by the user for a user interface object type in dialog box 240 will be output by the force feedback device by default, unless a different foreground application program deactivates the force sensations or replaces a force sensation with its own. For example, the user can specify that menu objects 242a in a GUI have a snap force 244a associated with them by moving the slider 246 to the right, where the slider scale specifies the magnitude of the snap force. Thus, when the user moves the cursor in the GUI over a menu item in a menu during normal operation, the user will feel a snap force, i.e., a force that draws the cursor/user object toward the center of the menu item to allow easier menu selection. This snap force is preferably applied to all menus of all running application programs in the GUI, since it is a background force effect. If the foreground application program has its own force sensations which define that application's menus to have a jolt instead of a snap, then the jolt will be superimposed on the snap unless the active application program deactivates the background force(s). In general, a particular active application program can only command forces to objects in its own windows, e.g., that application's own menus, scrollbars, icons, window borders, etc.

[0065] A user can specify multiple background force sensations 244 for each user interface object 242. This allows the multiple force sensations to be superimposed on each other, unless the application program overrides one or more of the superimposed forces. Thus, a user can assign a damper force sensation and a "ticks" force sensation to scrollbars in box 240, and all scrollbars will have these forces associated with them unless overriden by the foreground application program.

[0066] Other controls in box 240 include a device gain 248 to set the percentage of maximum magnitude for all the forces for items 242; and selections 249 to allow force feedback functionality on the host computer. Advanced button 250, when selected, preferably displays an additional window of options with which the user can customize the force sensations 244. For example, a user can specify positive or negative damping (where negative damping feels like moving on ice) or unidirectional or bidirectional dampers. The user can specify the spacing or orientation of snap points on a grid, the envelope parameters for a vibration, or the parameters for a snap force, e.g. the snap force is defined as an enclosure and the user can customize the enclosure to turn off forces on one wall, turn on inner walls, etc. Other application programs can also be used as the background or master application program 206.

[0067] Referring back to Figure 4, application programs 202, 204, and 206 communicate with an force feedback Application Programming Interface ("API") 208 which is resident in the host computer's memory and which allows a given application program to communicate with lower level drivers and other force feedback functions. For example, in the Windows operating system, API's are commonly used to allow a developer of an application program to call functions at a high level for use with the application program, and not worry about the low level details of actually implementing the function.

[0068] The API of the present invention includes a set of software "objects" that can be called to command a force feedback interface device 11. Objects are a set of instructions and/or data which can be called by a pointer and/or an identifier and parameters to implement a specified function. For example, three types of objects are provided in one preferred API implementation: interface objects, device objects, and effect objects. Each of these objects makes use of one or more force feedback device drivers which are implemented as objects in the API 208.

[0069] Interface objects represent the API at the highest level. An application program (which is referred to as a "client" of the API) can create an interface object to access lower level objects and code that implement force feedback device functionality. For example, the interface object allows an application program to enumerate and receive information about individual devices and create lower level objects for each device used by the application program.

[0070] Device objects each represent a physical force feedback device (or other compatible device or peripheral) in

communication with the host computer 18. For example, the force feedback mouse device 11 would be represented as a single device object. The device objects access the set of force feedback routines to receive information about an associated physical device, set the properties of the physical device, register event handlers (if events are implemented on the host), and to create effect objects.

[0071] Effect objects each represent a force feedback effect defined by the application program to be output one or more times on a force feedback device. The effect objects access the set of force feedback routines to download force effects to the force feedback device, start and stop the output of effects by the force feedback device, and modify the parameters of the effects. Event objects can also be created to define events, as described in greater detail below.

[0072] A force " effect," as referred to herein, is a single defined force or series of forces that may be commanded within the API. The effect typically has a name to identify it and one or more parameters to modify or customize the force as necessary. For example, several types of force effects have been defined, including vibrations, enclosures, grids, textures, walls, dampers, snap sensations, vibrations, circles, ellipses, etc. For example, a vibration force effect may have parameters of duration, frequency, magnitude, and direction. The force sensations 244 defined in dialog box 244 can be force effects; however, more generally, force sensations can be made up of one or more force effects. Force effects, in turn, are made up of one or more basic force prototypes, such as springs, dampers, and vector forces.

[0073] In the preferred embodiment, an application program client interacts with the API 206 by first receiving a pointer to a resident force feedback interface; for example, a preferred interface includes procedures provided in the Component Object Model (COM) interface of Microsoft Windows, an object oriented interface. Using the force feedback interface, the application program enumerates the force feedback devices on the computer system 18. The application program selects a desired one of the force feedback devices and creates a device object associated with that device. Using the force feedback interface, the application then acquires the device, sets the device up with setup and default parameters, and creates effect objects and event objects at times designated by the developer of the application. At appropriate times, the application program will command the start, stop, or pause of the playback of force effects by accessing the appropriate interface instructions associated with the desired effect. The API indicates to the context driver 210 to create a "context" (i.e. "effect set") for an application program, and sends effects and events to be associated with that context. A "context" is a group or set of effects and events that are associated with a particular application program.

[0074] Context driver 210 receives contexts 222 and device manipulation data 223 from the API 208. The context driver is provided at a level below the API to organize contexts for applications 202 and 204 running on the host computer. In the preferred embodiment, the effects and events in a context are not known to the application program itself; rather, the context driver 210 creates a context internally for an application. Thus, an application commands that a particular force sensation be output in response to different interactions or occurrences, e.g., an interaction of a cursor with an object or region, or the output of a force based on other criteria (time, received data, random event, etc.). The API sends the commanded effect(s) to the context driver 210, and the context driver stores the effects in the created context for that application program.

[0075] Since each application may have a completely different set of force effects to output, each context each must be associated with its particular application program. In the preferred embodiment, there are two types of contexts: foreground contexts and background or primary contexts. A foreground context is associated with the application program 202 or 204 that is active in the operating system. A background context includes the effects and events for the master application program 206, e.g., the effects and events necessary to implement those force sensations selected by the user in the dialog box 240 of Figure 5 to be associated with particular GUI objects. In addition, a "global context" can be provided, which includes common effects almost always used by application programs (e.g. a pop force) and which can automatically be downloaded to the force feedback device. Effects in the global context need not be stored in individual contexts for particular application programs and need not be sent to the force feedback device, thus saving memory on the device.

[0076] When an application program is first executed by the host computer and loaded into memory, if the application is able to command a force feedback device, the application will query for the API 208. Once communication is established, the API will contact the context driver 210 to create an entry location for a context set for the initiated application program.

[0077] The context driver 210 receives individual effects and events as they are created by the application program using API 208 and stores the effects and events in a context list 212, storing each context in a different storage location in the host's memory or on some other type of storage device. An active or inactive application program can create a context and have it stored, but only the active application's context will be sent to the force feedback device. The context driver 210 can examine an identifier in a created effect or event to determine which application program is associated with it and thus store the effect or event in the proper memory location. The context driver sets pointers to the contexts and to the effects and events in the contexts to access them. An effect can be created initially, before any forces or commanded, or the effect can be created and then immediately commanded to be output to the force feedback device. Each context also preferably includes an entry into a device state structure. This entry governs the "gain" or force level

for all effects in the context. For example, all the forces in the context can be cut to 50% of full magnitude by storing a value of 50 in the device state structure. One of the contexts stored in list 214 is a primary context 216, which is the list of effects and events used by the master application 206 or "background" application.

[0078] At a later time, after the context driver has stored the contexts in list 212, an application program may send a command to the API to output or " start" a particular force sensation. The API checks whether the application program is active or in the background (primary); if not, the API ignores the command. Alternatively, the commands from an inactive foreground application can be stored and then sent to the device when the application becomes active. If the application is active or background, the API sends the start information to the context driver 210 indicating the application program that commanded the force and the particular force effects to be commanded. The context driver 210 then associates the commanding application program with a context 214 in list 212 and sends the effects from the context to the force feedback device (if not previously sent). For example, if a context for a particular application program includes a spring effect, a damper effect, and a vibration effect, and the application program commands the vibration to be output, then the context driver selects the vibration effects to be output to the device. The data describing this effect is then output by the context driver 210. Similarly, the application program can sent a command to stop particular force effects, to pause the effects, to get the status information of an effect, or to destroy an effect.

[0079] A context is therefore only allowed to exert forces with the force feedback device when that context is active, i.e., is associated with an active application program or the background application. In the described embodiment, only one foreground context can be active at any given time. Any number of background contexts may be simultaneously active; however, there may be a device limit on the number of background contexts that may be created. For example, the mouse device 11 may only allow one background context to be created at any one time, which is the preferred embodiment. Thus, if an inactive (not in focus) foreground application program commands a force to be output, the API will ignore the command after determining that the commanding application is not active (or, the command will only be sent to the device when the application becomes active).

[0080] If the active application program becomes inactive (i.e. loses foreground or is removed from the host's memory) and a different application becomes active, then the API indicates this to the context driver 210, which then deactivates the context associated with that application program and loads the effects from the new active context to the force feedback device 11. Likewise, when the original application program again becomes active, the API tells the context driver to activate the associated context and load the appropriate effects to the force feedback device.

[0081] Device manipulation data 223 is also received by context driver 210. Data 223 is used to set a global device state on the force feedback device, as described below, and this information is passed unmodified to the translation layer.

[0082] Translation layer 218 manages the sending of effects to the device 11, receives information from the device to the host, and maintains a representation of the memory of device 11. Layer 218 receives an individual effect 219 for the active (or background) application program and device manipulation data 223 sent by the context driver 210. The translation layer receives the effect from an context list 220 of individual effects 222 (list 220 represents a context 214). A different context list 220 is provided in each context 214 of list 212. Each effect 222 in list 220 is a force that is to be output on the mouse 12 by the force feedback device 11. Then the effects are to be sent to the device 11 when commanded by the application, they are selected and copies are output to the device. Preferably, each effect is output by the translation layer as soon as possible, in the order of receiving the effects. Each effect stored in list 220 as examined by the translation layer is available on force feedback device 11, i.e., the local microprocessor should recognize the effect and be able to output the effect either immediately or when conditions dictate. The size of the list 220 should be the same or smaller than the available memory for such a list in the force feedback device so that the translation layer knows when the memory of the force feedback device is full. If full, the translation layer can delay the output of additional effects until enough memory space is available.

[0083] When an active application becomes inactive, the translation layer is instructed by the context driver 210 to "unload" the effects of the context of the previous active application from the force feedback device, receives the effects from the now active application and loads those effects to the force feedback device 11 (the effects for the background (primary) application are preferably never unloaded).

[0084] The translation layer also preferably handles events. For example, when a screen position is received from the device 11, the translation layer can check whether any of the conditions/triggers of the active events are met. If so, a message is sent which eventually reaches the associated active or background application program, as described in greater detail below. In alternate embodiments, the microprocessor 130 on device 11 can check for events and send event notifications through translation layer 218 up to the application program. However, this can be undesirable in some embodiments since the event notification is provided at a much slower rate than the force control loop of the microprocessor 130.

[0085] The translation layer also can store a device state 224 in memory. Device manipulation data 223 from the active application determines the device state. This is the state that the active application program wishes to impose on the device, if any. For example, an overall condition can be stored, such as an enable or disable for all forces, so

that if all forces are disabled, no forces will be output by the device. An overall gain can also be set to limit all output force magnitudes to a desired level or percentage of maximum output.

[0086] The translation layer outputs device messages 225 to the device 11 by way of the next layer. Messages may include force effects to be output and/or any other information such as device identification numbers or instructions from the context driver for an effect (start, stop, pause, reset, etc.) The translation layer outputs messages 225 in a form the device 11 can understand.

[0087] Device communication driver 226 communicates directly with the force feedback device. Driver 226 receives the device messages 225 from translation layer 218 and directly transmits the messages to the force feedback device over bus 120, e.g. a USB. The driver is implemented, in the preferred embodiment, as a standard driver to communicate over such a serial port of host computer 18. Other types of drivers and communication interfaces can be used in other embodiments.

Host Command Structures

[0088] The host computer 18 and API 208 handle a variety of processes controlling force feedback device 11. Different effects and events of the present invention used in force feedback systems are described below.

Effects

[0089] Effects are standardized forces that are determined according to a predefined characterization, which may include force algorithms, stored force magnitudes, functions of space and/or time, a history of stored motion values of the mouse, or other steps or parameters. An effect may be based on time and be independent of the position of the mouse 12, or may be spatially determined based on mouse position, velocity, acceleration, or any combination of these. Preferably, the device 11 includes a number of effects standardized in its memory which it can implement if the effects are within the active or background context. When the device 11 is determining output forces based on effects, the microprocessor 130 checks if the effect is active, calculates the raw contribution to the output force from the effect, applies an envelope scaling, and sums the scaled contribution to the total sum of forces contributed to by all effects currently being output. In determining the total sum, the device preferably combines all constant forces (e.g., conditions and time-based forces) and limits the constant force sum to a predetermined magnitude, then combines all dynamic forces and limits the dynamic force sum to a predetermined magnitude. The two sums are then added together and the total force sum is output by the actuators of the device 11. Alternatively, all forces can be treated the same and summed together.

[0090] FIGURES 6a and 6b illustrate one type of force effect provided by the host in a graphical environment such as a GUI and known as an "enclosure." An enclosure is at its most basic level a rectangular object provided in a GUI which may be assigned with different forces for each of its sides. The enclosure is a generic type of force feedback object that can be used for a variety of different graphical objects, including buttons, windows, sliders, scrollbars, menus, links on a web page, or grid points in a drawing program.

[0091] Figure 6a illustrates the use of an enclosure for a window in a GUI. The window 260 is displayed on the display device of the host computer. The enclosure has eight walls 262 associated with it invisible to the visual perception of the user, where four walls are provided as an inner rectangle 264 and four walls are provided as an outer rectangle 266. Each wall 262 can be separately defined as either providing a force pulling the user object toward the border of window 260 (attractive force), or pushing the user object away from the border of window 260 (resistive force). The outer rectangle 266 walls affect the user object when the user object is outside the window 260 between the rectangle 266 and the window 260 border, and the inner rectangle 264 walls affect the user object when it is inside the window 260 between the rectangle 264 and the window border. In the preferred embodiment, a window 260 is associated with a force enclosure by defining both outer rectangle 266 and inner rectangle 264 to have an attractive force, so that the user object and cursor tend to be attracted to the window border when within the appropriate range of the window. This, for example, allows the user to acquire the border and adjust the size of the window easily. Alternatively, the outer rectangle 266 can be defined as a resistive force (such as a spring) that requires the user to overcome the resistance to allow the user object entry into the window (and the rectangle 266 can also be defined as a click surface to cause a function to be applied when gaining entry). When the cursor/user object gains entry to an enclosure, an " event" is preferably indicated to the application program, as explained above. In an alternative embodiment, enclosures can similarly defined for circles, ellipses, or other shapes, both regular and irregular.

[0092] The enclosure object also preferably has a number of other parameters that further define the enclosure. For example, parameters may be specified separately for each wall to define such characteristics as the magnitude of the resistive or attractive forces, which of the eight walls are assigned forces, which walls provide clipping, the saturation level of the force applied in association with a wall, and the interior force, which can be specified as any force effect such as a damping, texture, or vibration force, that is applied while the cursor is inside the enclosure.

[0093] Figure 6b illustrates another embodiment of an enclosure. A button 270 can be a graphical button, a menu item, or an icon in a GUI. The outer rectangle 266 in this embodiment has been " turned off" so that no forces are associated with it. The inner rectangle 264 has been defined as a very small rectangle at the center of the button 270. Resistive forces are associated with the inner rectangular walls such that, once the user object has entered the enclosure, a force biases the user object toward the center of the button 270. This allows the user to target or "acquire" the button more easily with the cursor and reduces overshooting the button by accident by moving the cursor too far past the button.

[0094] Enclosures are quite useful for adding forces to web links on a web page viewed in a web browser, such as Netscape Communicator by Netscape Communications or Internet Explorer by Microsoft. The inner wall forces of Figure 6b can be used on a web link displayed on a web page to draw the cursor/user object onto the link, thus allowing a user to more easily determine which images are links on the page. Other types of forces can also serve this function. Also, an " inside condition" such as a texture can be provided for the web link to cause some links to feel rough, some smooth, some bumpy, etc.

[0095] An example of an enclosure effect structure is provided below:

```
ENCLOSURE
typedef struct {
DWORD dwSize;
        RECT rectOutside;
        DWORD dwXWidth;
        DWORD dwYWidth;
        DWORD dwXStiffness;
        DWORD dwYStiffness;
        DWORD dwStiffnessMask;
        DWORD dwClippingMask;
        DWORD dwXSaturation;
        DWORD dwYSaturation;
        LP_CONDITION pInsideCondition
} ENCLOSURE;
```

dwSize: Size of the structure in bytes.

rectOutside: Outside borders of the enclosure, in screen coordinates. When the mouse position is outside of rectangle defined by rectOutside, there are no forces exerted on the device from this enclosure.

dwXWidth, dwYWidth: Width W of the region between a wall and the object border. In one implementation, the width of the inner wall and the outer wall regions is the same.

Alternatively, they can be defined separately. This parameter can alternatively be specified as a deadband.

dwXStiffness: Stiffness of the left and right walls. Defines the amount of horizontal resistance (force) felt by the user when attempting to break through a vertical wall. This value amounts to the negative and positive coefficient of horizontal spring conditions that are applied when the mouse is in the left and right walls, respectively.

dwYStiffness: Stiffness of the top and bottom walls. Defines the amount of vertical resistance (force) felt by the user when attempting to break through a horizontal wall.

dwStiffnessMask: Mask specifying when resistance should be applied. Can be one or more of the following values:

NONE: No resistance is felt on any wall.

INBOUNDTOPWALL, INBOUNDLEFTWALL, etc.: Resistance is felt when entering the enclosure through the top, bottom, left, or right wall, as specified.

OUTBOUNDTOPWALL, OUTBOUNDLEFTWALL, etc.: Resistance is felt when exiting the enclosure through the top, bottom, left, or right wall, as specified.

INBOUNDANYWALL, OUTBOUNDANYWALL: Resistance is felt when entering or exiting, respectively, the enclosure through any wall.

ANYWALL: Resistance is felt when entering or exiting the enclosure through any wall.

dwClippingMask: Mask specifying when mouse cursor clipping should be applied. Can be one or more of the following values:

NONE: Clipping is not applied to any wall

INBOUNDTOPWALL, INBOUNDLEFTWALL, etc.: Clipping is applied when entering the enclosure through the top,

bottom, left, or right wall, respectively.

OUTBOUNDTOPWALL, OUTBOUNDLEFTWALL, etc.: Clipping is applied when exiting the enclosure through the top, bottom, left, or right wall, respectively.

INBOUNDANYWALL, OUTBOUNDANYWALL: Clipping is applied when entering or exiting, respectively, the enclosure through any wall

ANYWALL: Clipping is applied when entering or exiting the enclosure through any wall.

dwXSaturation, dwYSaturation: Saturation level of spring effect felt by the user when attempting to break through a vertical or horizontal wall, respectively.

pInsideCondition: Points to a condition that is to be in effect when the user mouse is in the deadband area. For example, can specify a spring condition, damper condition, and/or texture condition inside the enclosure. This is useful in graphical environments such as GUI's and web pages/ web browsers.

[0096] In another example of an enclosure, a circular or elliptical shape is defined. For example, an elliptical enclosure can include only one wall that is curved all the way around the ellipse. The inner and outer walls can still be provided, as well as the rest of the other parameters. The dimensions of the ellipse/circle can be provided as X and Y widths. Alternatively, an eccentricity parameter can define the eccentricity of the ellipse.

[0097] One problem with enclosures is that often the forces of the enclosure are desirable at one point in time but not at a different point in time. For example, a button having an enclosure has a attractive force associated with the enclosure to assist the user in acquiring the button with the cursor. However, once the cursor is positioned inside the enclosure, the user may want to be able to freely move the cursor again without enclosure wall forces, since the target button has already been acquired. Thus, in one embodiment, the forces associated with the enclosure are turned off once the cursor is moved inside the enclosure. Alternatively, the force magnitudes can be weakened or adjusted to a lower level, e.g. 1/3 the normal magnitude, to allow the cursor to easily exit the enclosure. Once the cursor has exited, the forces can be turned back on. In an alternate embodiment, the forces associated with the enclosure can be turned off or weakened for a predetermined period of time to allow the user to move the cursor out of the enclosure without hindrance, and then the forces can be automatically turned back on once the time has expired. I yet a different embodiment, the forces are turned off only when the cursor has not moved for a predetermined period of time; this indicates that the user has acquired the desired target. In still a further embodiment, the forces can be turned off when a command gesture, such as a button 15 activation, is detected by the host or microprocessor 130. These various embodiments can also be combined in other embodiments as desired. The host or microprocessor can determine when the enclosure has been entered and exited by the cursor through the use of events, explained below.

[0098] Enclosures can also be defined to compensate for user controlling peculiarities. For example, it has been found that up-down movement of the mouse 12 for most users is much easier that left-to-right movement. An enclosure having equal strength forces on all walls will feel as if the left and right walls have stronger forces. Thus, the upper and lower walls can be defined with stronger forces, e.g., two times the magnitude of the left and right wall forces, to compensate for this effect. Such an enclosure typically feels as if forces of all the walls are of equal strength.

[0099] FIGURE 6c illustrates another use of enclosures in the present invention. Enclosures can be used with events to provide some enclosures with priority over other enclosures. The enclosure with priority can change the characteristics of another enclosure, such as the strength of forces of the walls of the other enclosure. For example, window 272 is associated with a force enclosure 274. The walls of enclosure 274 surround the borders 275 of the window and have an associated force to attract the cursor/user object to the border. Buttons 277 are also included in window 272 which are near the border 275 and which may be selected by the user. Buttons 277 have their own enclosures defined in the interior of the buttons. The problem is that the forces from the enclosure 274 and the forces from the button enclosures conflict. A user may not be able to select the button easily due to the cursor being stuck between competing enclosure forces.

[0100] Preferably, an enclosure 276 can be defined to control the effects of other enclosures. For example, the enclosure 276 occupies a region around buttons 277 and is preferably not visible to the user. Enclosure 276 turns off the forces of enclosure 274 inside its borders and leaves the snap forces of the buttons 277 on. This allows the user to feel the snap forces of the buttons without conflict from the forces of the window enclosure 274. Alternatively, the forces of enclosure 274 can be reduced inside enclosure 276 to an extent so that the user is not encumbered by competing forces. A similar type of enclosure can be user on sliders, where the moveable "thumb" of the slider can be provided with an enclosure similar to enclosure 276 surrounding it. The enclosure surrounding the slider turns off or reduces forces from other enclosures and allows the snap attractive force of the thumb to attract the user object/cursor to the thumb. Events can be used to determine when the cursor enters enclosure 276.

[0101] FIGURE 7 illustrates a different effect useful in force feedback applications known as a texture. A texture is a condition, similar to friction, which causes the mouse to feel as if it were traveling over a series of bumps. Figure 7 illustrates a distance vs. force graph in which a number of bumps are defined with a coefficient C, a period T, and a deadband DB. Bumps in the positive distance region are felt when the mouse 12 is moved in a positive direction, and bumps in the negative distance region are felt when the mouse 12 is moved in the negative direction. There is a separate

height (coefficient) and period of the bumps for the positive and negative directions along a particular axis of the mouse 12 so that different textures can be felt depending on the direction of the mouse. Directional textures can be used, for example, to achieve a velvet-like effect. The deadband value can apply for both directions. Textures are defined as a condition and can include parameters such as:

Positive Coefficient, Negative Coefficient: Height (magnitude) of bumps when travelling along the axis in the positive or negative direction, respectively.
Positive Saturation, Negative Saturation: Period, in pixels, of bumps (bump width plus deadband width) when travelling along the axis in the positive or negative direction, respectively.
DeadBand: Percentage of period which is not occupied by a bump.

[0102] FIGURE 8 illustrates a different force feedback effect useful in force feedback applications known as a grid. A grid is a condition that creates a 2-dimensional array of snap points 280, which are forces attracting the user object to the point when the user object moves over the point. Grids can be stand-alone conditions, but they can also be useful when associated with enclosures. The geometrical attributes of a grid inside of an enclosure 282 are depicted in Figure 8. Grids are defined as a condition and can include parameters such as those below, also illustrated in Figure 8.
Offset (O): If the grid is inside of an enclosure, defines the horizontal and vertical distance, in pixels, from the upper left comer of the enclosure's deadband to the first snap point. If the grid is stand-alone, defines the horizontal and vertical distance, in pixels, from the upper left comer of screen to the first snap point.
Positive Coefficient (C) : Strength of snap points (jolts).
PositiveSaturation (S): Saturation of snap points.
DeadBand (DB): Vertical and horizontal spacing, in pixels, between snap points.

[0103] Grids can also be implemented as one-dimensional forces, e.g., as tick marks on a slider or any other graphical object when moved. The tick marks would only be felt when the mouse was moving in the defined dimension. For example, a slider can be defined with a one-dimensional grid so that, as the knob of the slider is moved, a snap point is applied as the knob moves over each tick of the grid. The ticks thus have a spatial correlation with the distance that the knob has moved and inform the user of such.

Events

[0104] One structure used by the host in force feedback implementation is called an "event." Events are notifications from the force feedback device to application programs 202, 204, and 206 of one or more actions or interactions that have occurred in the graphical environment which has been sensed by the force feedback device 11. For example, an event can be any interaction of the cursor in the graphical environment with another object. Thus, an event can be defined when the cursor enters into an enclosure object (described below) through a particular border. Alternatively, an event can be triggered when the cursor moves over a close box of a window, or the cursor moves over a file icon and a button is pressed. An event can also be defined when the cursor moves within a specified range of a particular graphical object or region. Also, an event may be defined with a temporal component, e.g. an event is triggered when the cursor remains in a particular region for a specified period of time. Events are similar to button press triggers in that a condition is detected on the force feedback device, independently of the host; however, the trigger for an event is a graphical interaction rather than a physical button press.

[0105] An application program initially defines what the event is by creating an event definition with the API 208. The event definition preferably includes an event identifier, an application program (or window) identifier, information specifying actions that trigger an event notification, and force effects, if any, that the event is associated with. The application program assigns a separate identification value to each event at the time of creating the event, and keeps track of the created identification values in its own list. The API can enable, disable, or delete an event at the request of an application program.

[0106] The translation layer receives and stores events when the application that created the event is the active or background application program. Thus, events are preferably included in the active context. The translation layer preferably checks for events based on the position of the cursor (or mouse) received from the force feedback device when the associated application program is (or becomes) active. The translation layer then sends the event notification up to the context driver (or API), which sends an event notification to the application program, e.g. sends a message to a window handler which the application program checks. The event notification includes the event identifier so that the application program can check its own list and identify the event. Once identified, the application program initiates the appropriate response as defined by the developer, e.g. running another application program, outputting a sound, displaying a new image, etc. Alternatively, the force feedback device is loaded with the defined event similar to being loaded with effects. The local microprocessor 130 can check for events (for active application programs) and send information back through the translation layer to the context driver to notify the application program.

[0107] For an application program to receive an event notification, the application's context must be active, i.e., the application must be active or background. Thus, inactive applications will not receive event notifications. Some events can be defined to have priority over other events. For example, if a trigger is associated with an event in a foreground context as well as an event in a background context, the foreground context can be defined to have priority so that only the application program in the foreground receives an event notification. Alternatively, only the background application, or both the foreground and background applications, can receive the event notification.

[0108] Preferably, both one-time event triggers and periodic event triggers are available. One time event triggers are triggers associated with discrete actions that have no significant duration, such as breaking through an enclosure. Periodic event triggers are associated with conditions that have a finite duration, such as being inside the wall of an enclosure, or holding down a scroll button. During the duration of the condition which triggers the event (e.g., while the cursor is inside the enclosure), the API repeatedly sends event notifications to the application program according to a predetermined periodicity that can be set by the application program.

[0109] As an example, an application program creates an event definition for an enclosure event. This is an event that occurs when a user moves the cursor into an enclosure. The event definition includes a unique event identifier, an identifier for the application program that created it, actions that trigger the event notification (entering the enclosure), and any effects (such as the particular enclosure) which the event is associated with.

[0110] An example of an event structure is provided below:

```
EVENT
typedef struct {
  DWORD          dwSize;
      HWND       hWndEventHandler;
      WORD       wRef;
      DWORD      dwfTrigger;
      LPI_EFFECT pEffect;

} EVENT;
```

dwSize : Size of the structure in bytes. This member must be initialized before the structure is used.

hWndEventHandler: Handle of window to which event notifications are sent.

wRef: 16-bit application-defined value to identify the event. This value is defined by the application to identify events that it has created. The application assigns a separate wRef value to each event at the time of creation. Each subsequent event notification that the application receives will be tagged with its associated wRef value. When the application processes the window message that receives event notifications, it will compare the wRef value to it's own internal list and take action accordingly.

dwfTrigger: Flags specifying actions that trigger the event notification.

pEffect: Particular effect (e.g. enclosure), if any, in the GUI that this event is associated with.

The following are examples of flags that can be included as triggers for an event.

ONENTERTOP, ONENTERLEFT, etc.: The mouse broke into the associated enclosure by breaking through the top, bottom, left, or right wall, as specified.

ONEXITTOP, ONEXITLEFT, etc.: The mouse broke out of the associated enclosure by breaking through the top, bottom, left, or right wall, as specified.

ONENTERANY, ONEXITANY: The mouse broke into or out of the associated enclosure.

INBOUNDTOPWALL, INBOUNDLEFTWALL, etc.: The mouse is in the top, bottom, left, or right wall, as specified, and attempting to break into the associated enclosure.

OUTBOUNDTOPWAL, OUTBOUNDLEFTWALL, etc.: The mouse is in the top, bottom, left, or right wall, as specified, and attempting to break out of the associated enclosure.

INBOUNDANYWALL, OUTBOUNDANYWALL: The mouse is in a wall and attempting to break into or out of the associated enclosure.

ANYWALL: The mouse is in a wall of the associated enclosure.

ONSCROLL: The user is holding down the scroll button of the mouse.

ONEFFECTCOMPLETION: The associated effect has completed.

[0111] The host computer or force feedback device may also command the mouse 12 to act as a different type of control device for some embodiments. The force feedback device 11 is primarily intended, in the embodiment described

in Figure 2, to be used in a graphical environment such as a GUI, in which multi-tasking applications are provided. When the force feedback device is used as such, the API 208 and other layers described above are preferably used. However, the force feedback device can also be used as a different type of controller and may use other standard API's. For example, the mouse force feedback device 11 as shown in Figure 2 may also be desired to be used in conjunction with game application programs, simulations, or the like, in which a joystick or other type of controller is typically used. Device 11 can be configured to work with such game applications in a "joystick mode" as if it were a standard force feedback joystick, or as if it were a non-force-feedback joystick. The user can be provided with a physical switch on the device 11 to switch from mouse to joystick mode, or a software switch provided in master application 206 or other program on the host computer can be used to switch modes. For example, when the device is in the joystick mode, the host computer can use the DirectX API from Microsoft Corp. which includes force feedback functionality for standard joystick controllers. The device 11 becomes an absolute position reporting device in joystick mode.

[0112] Mouse device 11 as shown in Figure 2 is not easily applicable to many applications intended for use with a joystick, since the mouse is a planar device having a free workspace while a joystick typically has rotary degrees of freedom and springs to bias the joystick to a center position. Thus, forces are preferably used to cause the mouse to be more similar to a joystick. When the mouse device 11 is used in the joystick mode, in the preferred embodiment, spring forces are applied to bias the mouse 12 toward the center of its workspace. These forces are applied by actuators 64 and controlled by the local microprocessor 130 as background forces that are always in effect, unless the device is switched out of joystick mode.

Force Feedback Device Implementations

[0113] The force feedback interface device 11, as described above, preferably includes a local microprocessor 130 that implements various processes to provide the functionality and features of the force feedback implementation described herein. Various aspects of the force feedback device implementation are described below.

Reference Frames

[0114] The force feedback mouse device as described herein needs the ability to know the location of the cursor on the display device of the host computer. This is because the device must monitor the interaction of the cursor with other objects in the graphical environment to determine when associated forces should be output. The force feedback device of the present invention uses different reference frames to determine where the cursor is positioned. There are three primary reference frames that are described herein: the device (or local) frame 30, the ballistic frame and the screen (or host) frame 28.

[0115] The device frame is the frame of the physical workspace of the force feedback device. In a preferred embodiment, the device frame is defined with the origin at the top left, the positive x direction to the right and the positive y direction down. This frame is used to describe the absolute position of the end point of the mechanism in its workspace. The scale of "movement units" for the device frame is arbitrary and fixed, although it should be a higher resolution than the resolution of sensors 62 to guarantee no loss of sensor data. Furthermore, in the preferred embodiment, the device position is scaled to reflect a 4:3 aspect ratio, which matches the preferred mechanical design (guide opening 76) as well as the typical computer monitor size. However, in designs having different-sized mechanical workspace, the device frame is scaled to the provided mechanical workspace.

[0116] In the preferred embodiment, rate-control indexing is provided. This allows the mouse 12 to reach the limit to its workspace and still control the cursor in the same direction. In brief, the central portion of the device workspace is designated as the position control region, where a change in position of the mouse in its workspace directly correlates to a change in position of the cursor on the screen. The change in device position is translated to a change in ballistic position based on the velocity of the device, according to a ballistics algorithm (described below).

[0117] An edge region of the device frame is designated as the rate-control region, in which the mode of moving the cursor changes from position control, to rate control, where the cursor continues to move on the screen in the same direction while the mouse is in the rate control region. The rate control region boundary is accompanied by a repelling force that pushes the user object towards the center of the device workspace. As the user pushes against this force the cursor is moved in a speed proportional to the distance the device is displaced into the rate control region. Therefore, the cursor moves relative to device position in the rate control region, not relative to device motion. This mode may be thought of as moving the position control central area of the device frame around the screen using rate control. The size of the rate control region is specified by a percentage. For example, a 400 x 300 device workspace has a rate-control region of 10%. The cursor position would be determined according to a position control paradigm when the user object position is read between the values of (x,y) = (20,15) and (380, 285) of the device frame, and according to a rate control paradigm if the user object position is outside of that range.

[0118] The screen frame is used to describe the position of the cursor (or other user-controlled graphical object) on

the screen. It is defined with the origin at the top left, the positive x direction to the right and the positive y direction down. This frame is scaled corresponding to the resolution of the display device 20 (or devices 20). For example, if a computer monitor resolution is set to 1024 x 768 pixels, the x screen position ranges from 0 to 1023 as the cursor moves to the right and from 0 to 767 as the cursor moves down. All coordinate communication between the force feedback device and the host computer are in terms of screen coordinates (normalized for device standards).

[0119] The ballistic frame (or "scaled frame") is a frame created by the force feedback device to determine cursor position. By applying a ballistics algorithm to the change in position of the user object, the cursor position is determined. The ballistic frame is defined as a multiple of the screen frame, i.e., the ballistic frame preferably has a higher resolution than the screen frame, but has the same aspect ratio as the screen frame. The ballistic frame is created by the force feedback device so that pixel changes smaller than one screen pixel can be made to the cursor position, which is crucial when implementing the fast control loop rates demanded by force-feedback. For example, if the control frequency is 1kHz, and a monitor provides 1000 pixels across, a change of 1 pixel per control cycle would cause the cursor to travel the entire screen width in one second. In rate control mode, such adjustments in cursor position may have to be made every control cycle, which is far too fast to allow control of the cursor. Thus, a frame with higher resolution, the ballistic frame, needs to be defined to allow the force feedback device to change cursor position at its own control frequency, but which can be divided down to the screen frame size to slow down the cursor as it is moved across the screen. In addition, the aspect ratio of the ballistic frame is the same as that of the screen frame. This allows force effects on the device to spatially correlate with graphical objects displayed on the display device. This is described in greater detail below.

[0120] The microprocessor must make transformations between frames to convert a device position to an eventual screen coordinate. When transforming from the device frame to the ballistic frame, the microprocessor may use sensitivity. Sensitivity is the ratio of the size of the device frame to the size of the ballistic frame. A higher value of sensitivity allows the cursor to traverse the screen with less physical motion of the mouse 12, thus causing the cursor to be harder to control. A lower value of sensitivity allows more motion on the device to cause less motion of the cursor on the screen, and is suitable for fine cursor control. At low sensitivity values, the cursor may only be able to reach the entire screen area by using rate control mode.

[0121] In the preferred embodiment, the cursor can exactly move across the entire width of the screen as the mouse 12 moves across the extent of the position control region of the device workspace, as long as the velocity of the mouse 12 remains in a middle range to provide a one-to-two ballistic mapping (a one-to-one ballistic mapping is provided with velocities in the lower range, causing less cursor movement; and a one-to-four ballistic mapping is provided with velocities in the high range, causing more cursor movement). As sensitivity is increased, the screen can be traversed in less than the entire position control region of the device workspace (assuming a middle range velocity). At lower sensitivities, the entire screen cannot be traversed (assuming a middle range velocity) without using the rate control mode.

[0122] Since the aspect ratios of the device frame and the ballistic frame may not match, updating position changes in the ballistic frame may require that a change in device position (or a simulated change, when in rate control mode) be added to the ballistic frame. If changes in mouse position are always used, problems with different aspect ratios are eliminated. This is described in greater detail below.

[0123] Transforming from the ballistic frame to the screen frame is simpler. Once the ballistic cursor position is known, the cursor position in the screen frame can be found by dividing the ballistic cursor position by the known multiple. Thus, if the ballistic frame is defined to be 10 times the resolution of the screen frame, the ballistic position is divided by 10 to obtain the screen position.

Position Reporting

[0124] Standard mouse controllers are open-loop devices which report position changes of the mouse in its workspace to the host computer and have no ability to know the current cursor position. The force feedback mouse device of the present invention, in contrast, needs to know the location of the cursor on the display device of the host computer, since the mouse device must monitor the interaction of the cursor with other objects in the graphical environment to determine when associated forces should be output.

[0125] There are two primary means for a force feedback mouse to report the position of the mouse to the host and to know the screen position of the cursor. The first, preferred method for the mouse to know cursor location is for the mouse to report (normalized) absolute coordinates to the host computer to dictate the position of the cursor. For example, the actual screen coordinates of cursor can be determined by the mouse and reported to the host, and the host simply displays the cursor at the reported coordinates. This allows the mouse to know exactly where the cursor is at all times, and to therefore have an accurate model of the screen for use with outputting forces. However, any ballistic behavior of the cursor must be calculated on the mouse device itself, creating more calculation overhead for the local microprocessor 130. In addition, an absolute reporting scheme is not compatible with traditional, non-force-feedback mouse devices, thus excluding a default mode in case the force feedback functionality is not operational.

[0126] A second method provides a force feedback mouse device that reports position changes of the mouse in its workspace, as a standard mouse does today, rely on the host computer to send back the cursor position. This approach allows the host software to maintain the job of doing cursor ballistics, but requires twice the communication as well as a need for the mouse to predict intermediate values of cursor position while waiting for the next host cursor position update, since the cursor reporting occurs at a slower rate (e.g. about 50-60Hz) than the haptic control rate (e.g. about 1000Hz). This embodiment is described in greater detail with respect to Figure 13.

[0127] The processes described below are preferably implemented as firmware in integrated circuits and memory provided on the force feedback device. Alternatively, the processes can be implemented as hardware or software, or a mixture of these. The processes themselves can be implemented using program instructions or code stored on or transferred through a computer readable medium, such as a memory chip, circuit or other device; magnetic media such as hard disk, floppy disk, or tape; or other media such as CD-ROM, DVD, PCMCIA cards, etc. The program instructions may also be transmitted through a channel to device 11 from a different source.

[0128] FIGURE 9 is a block diagram illustrating a first, preferred embodiment 290 of the present invention. In this embodiment, both relative and absolute position reporting modes are used. As referred to herein, relative position reporting means that the force feedback device reports only changes in position of the user object 12 to the host computer, i.e. a difference between a new position and the last reported position, while absolute position reporting means that absolute coordinates are reported to the host from which the host can directly display the cursor or other graphical object. Most force feedback controllers use absolute position reporting since the position of the cursor on the screen must be known to implement forces.

[0129] One problem with absolute position reporting is that the host computer cannot detect the force feedback device as a standard input controller, such as a mouse. If relative position reporting were provided, the host computer can detect the force feedback mouse as a traditional non-force-feedback mouse. This is desirable when force feedback functionality is not currently active to allow the user to select options within the operating system, such as during startup of the device before force feedback drivers have been loaded or during emergencies when force feedback is not enabled.

[0130] The present embodiment provides a relative positioning mode when the force feedback device 11 is powered up or when the host computer 18 first detects the force feedback device. In this mode, the device 11 provides a device delta position to the host as shown in Figure 9. The host is expecting to receive a delta value upon startup, and thus simply processes the delta position normally. This allows the user to use the device for normal postioning of the cursor in a GUI or graphical environment before force functionality has been initialized. The microprocessor does not need to model the cursor position in this stage of the process because no force feedback functionality has yet been enabled on the device 11. However, once the force feedback functionality is enabled, e.g., the force feedback driver software has been enabled on the host computer, then an absolute position is reported to the host in place of the delta position relative position reporting. In addition, ballistic parameters and the host screen size are sent to the device 11 to allow the device 11 to determined the absolute cursor position. Force feedback commands are also sent to the device in this stage when appropriate.

[0131] FIGURE 10 illustrates a process 300 for providing force feedback according to the preferred embodiment 290 of Figure 9. In a first branch of process 300, current joint angles 302 are measured by sensors of the force feedback device 11. For example, in the embodiment of Figures 1 and 2, the joint angles between members 44, 46, 48, and 50 are determined based on the current position of the encoder arc as sensed by the emitter and detector of sensor 62. The joint angles can be determined since the precise measurements of the members of linkage 40 are known and are in a known position relative to each other based on the sensor 62 reading. In a process step 304, the current joint angles are normalized based on the sensed range of the mouse 12 in its degrees of freedom. Since the exact range of the mouse 12 (or other user object) may not be precisely known due to variations in the device or other irregularities, the normalization process uses a sensed range of motion to determine and assign physical angles between the members of the linkage 40. For example, the minimum sensor value that has been detected so far is considered to be one limit of the motion of the mouse, and the maximum sensor value detected so far is considered to be the other limit to the mouse range. In successive iterations of the process 300, the mouse eventually will be moved to the physical limits of the device and the angle values will be updated accordingly. This tends to produce more accurate results than an assumed range of motion, since manufacturing tolerances of the components of device 11 provide slightly varying dimensions, link lengths, and angle ranges that cause discrepancies from device to device. In an alternate embodiment, the force feedback device (microprocessor 130) can control the actuators of the device to move the mouse 12 to all the limits of the mouse 12 in a short start-up procedure to determine the actual angle ranges.

[0132] The normalized joint angles 306 resulting from step 304 are then processed by applying kinematics in step 308. Kinematic equations, as is well known to those skilled in the art, are used to derive a position of a desired point on a mechanical linkage or other apparatus from known lengths of links in the linkage and the current angles between those links. For example, a cartesian position of a point on the linkage in a cartesian (or other type) of coordinate system can be determined using geometrical equations. A current device position 310 is determined from the kinematics

of step 308, which is the position of the mouse 12 in its workspace.

[0133] The current device position 310 is used, with a previous device position 312, to determine a device delta position 314. The previous device position 312 was the current device position 310 in a previous iteration of process 300; this is indicated by the dashed line 311 between current and previous device positions. The device delta position 314 is simply the difference between the current device position 310 and the previous device position 312. In step 316, the microprocessor 130 sends the determined delta position 314 to the host computer 18 over bus 120. This is a relative position reporting step and is only performed if the system 10 does not yet have force feedback functionality, e.g. at startup when the force feedback drivers are being loaded. The host computer uses the delta position to position the cursor on the screen of display device 20. For example, the host computer adjusts the position of a cursor by the delta amount to achieve a new position of the cursor in the screen frame 28. During relative position reporting mode, the delta position 314 is sent to the host at a slower rate than the rate of process 300; therefore, in some iterations of process 300, the delta need not be reported to the host. This is because the host only needs to receive a cursor delta position at the rate of displaying the cursor on the screen, which typically is on the order of 60 Hz or every 20 ms. The microprocessor 130, on the other hand, must control forces at a much higher rate (1000 Hz or above) and thus needs to execute process 300 much faster.

[0134] If the device is in relative position reporting mode, the process iteration is complete after the delta position 314 is reported to the host. The process then returns to measure joint angles 302 and start the next iteration when triggered to do so. The microprocessor does not need to model the cursor position in this mode because no force feedback functionality has yet been enabled on the device 11.

[0135] Relative position reporting mode is exited and absolute position reporting mode is entered when the appropriate software drivers or other programs are loaded into the memory of host computer 18 to allow the host to recognize, interface with, and command the force feedback of device 11. In absolute position reporting mode, the delta position 314 continues to be determined as described above. However, the delta position is no longer reported directly to the host computer, but is instead used in a ballistics process to determine a ballistic delta, described below.

[0136] In absolute position reporting mode, the force feedback device 11 also determines a velocity of the user object 12 for use in the present process in addition to the joint angle processing described in steps 304 and 308. A current joint velocity 320 is first measured. In the preferred embodiment, this is accomplished by a dedicated circuit or device that determines a velocity from the sensed positions reported by sensors 62, e.g. the sensed positions of members 44 and 48. For example, a "haptic accelerator" device can include a circuit dedicated to calculating velocity from multiple sensed positions and time periods between the sensed positions. Alternatively, velocity sensors can be used to directly measure velocity of joints or members of the device.

[0137] The current joint velocity 320 is used in a process step 322 which uses kinematics in determining device velocity. As described above, kinematics are well-known equations for determining position, and also are used to determine a velocity of a desired point of a linkage. Thus, the kinematics can be used to determine the velocity of the mouse 12, which is referred to as the "device velocity" herein. The current device velocity 324 resulting from step 322 indicates the current velocity of mouse 12.

[0138] Ballistics step 328 uses a ballistic algorithm or similar method to model or predict a position of the cursor in the screen frame 28 based on the position and velocity of the mouse 12. This modelling is accomplished to allow the local microprocessor 130 to determine where the cursor is positioned in the ballistic frame. Using the spatial layout of graphical objects that is stored locally in memory 134 and which is continually updated by the host, the local microprocessor determines when the cursor is interacting with an object and when to output forces associated with the interaction. The local determination of the cursor position allows the microprocessor to know the position of the cursor for the determination of forces.

[0139] In the ballistics step 328, the local microprocessor uses a ballistics algorithm to determine a "ballistic delta" for the cursor, i.e. a change in position of the cursor measured in the ballistic frame. A ballistics algorithm is a method of changing the position of the cursor so as to break the direct mapping between the mouse 12 position and the cursor position to allow the user greater control over the cursor position. For example, most ballistics algorithms vary the position of the cursor based on the velocity of a mouse in its workspace. The faster the velocity of the mouse, the greater the distance that the cursor is moved on the screen for a given distance the mouse is moved. This allows slow motions of the mouse to finely control the cursor, while fast motions coarsely control the cursor and allow the cursor to be moved quickly across the screen. Other algorithms can also be used which are similar to ballistics in that cursor position is altered to allow fine cursor control in situations when needed (such as for targeting graphical objects with the cursor) and to allow coarse cursor control when needed.

[0140] The microprocessor 130 preferably uses a ballistic algorithm that provides two velocity thresholds to determine how far the cursor is to be moved. This is described in greater detail below in the method of Figure 11. Other ballistics algorithms may be used in other embodiments; e.g., a continuous algorithm that adjusts cursor position continuously along a range of mouse velocity.

[0141] The microprocessor uses several types of data to determine the ballistic delta of the cursor, including the

device delta position 314, current device velocity 324, and host ballistic parameters 330. The host ballistic parameters 330 are received by the microprocessor 130 from the host at some point before the process 300 starts, or when the ballistic model changes, and are stored in local memory 134 to be accessed when required. Such parameters can include the distance to move the cursor for each threshold range of device velocity, the velocity values of the two thresholds, and any conditions concerning when to apply the ballistics algorithm. If the host changes ballistics algorithms, notification can be sent to the microprocessor 130 and new parameters stored in local memory 134. The microprocessor 130 examines the current device velocity 324 and the parameters 330 to determine in which threshold range the mouse velocity currently is placed, and then multiplies the device delta position by the appropriate amount (e.g., 1, 2, or 4, depending on the velocity). This results in the ballistic delta 332, which is provided in terms of the ballistic frame. Changes in mouse position (i.e., delta positions 314) are used in the absolute position reporting mode of the present invention because this allows problems with changes to screen aspect ratios and resolutions to be alleviated. In addition, an indexing function of the device 11 is preferably implemented which may cause the ballistic delta 332 to be calculated differently in a rate control mode, as detailed below in Figure 11.

[0142] The microprocessor then proceeds to a process step 334, in which a current ballistic location (or "scaled position") 336 of the cursor is determined in the ballistic frame. To accomplish this, the microprocessor uses the ballistic delta 332, and the previous ballistic location 338. The previous ballistic location 338 was the current ballistic location 336 in an earlier iteration of process 300, as indicated by dashed line 337. Previous location 338 can be stored in local memory 134 and then discarded after being used to determine the current ballistic location. At its simplest level, the current ballistic location 336 is determined in step 334 by taking the previous ballistic location 338 and adjusting that location by the ballistic delta 332.

[0143] The current ballistic location 336, however, cannot be directly transformed to a screen pixel position in the screen frame. Step 342 uses the current ballistic location 336 and the host screen size 340 to determine the screen pixel location.

[0144] The host screen size 340 is the current pixel resolution and/or aspect ratio being displayed on the display device 20 of the host computer 18. For example, the host may be displaying a screen resolution of 800 x 600 pixels, 1024 x 768 pixels, or 1280 x 960 pixels. Also, in some embodiments, the host may be displaying a different aspect ratio. For example, a typical computer monitor aspect ratio is 4:3; however, if two monitors are connected to provide a continuous screen space, the aspect ratio becomes 8:3. i.e., double the size of a single screen in one dimension. The host screen size 340 can be received by the device 11 when the ballistic parameters 330 are received, or at any other time (and/or when sensitivity information is received, if applicable). Different resolutions or aspect ratios have different numbers of pixels in respective dimensions of the screen and thus cause displayed objects to appear in different areas of the screen. For example, if a graphical object is centered in the screen at 640 x 480 resolution, the object will appear in the upper left quadrant of the screen when the resolution changes to 1280 x 960.

[0145] The change in resolution or aspect ratio can create a problem with force sensations created by the force feedback device. The device has been sent a spatial layout of objects from the host that indicates the coordinates of an object on the screen. When the screen size is changed, and the device is not informed, the device will output forces corresponding to the object as it appeared in the old screen size. This can cause large discrepancies between where an object is displayed and where the forces of an object are encountered in the device workspace. For example, if an icon is at coordinates (64,48) on a 640x480 resolution screen, the device knows that the icon is positioned 10% of the entire screen dimension away from the top and left borders of the screen workspace. The device accordingly outputs associated forces when the mouse is moved into the region of the device workspace corresponding to the icon, i.e. 10% of the workspace distance from the top and left borders of the workspace. However, after a resolution change to 1280x960, the icon is displayed 5% of the screen dimension away from the top and left borders. The device, however, would output forces based on the old position of the icon, which no longer corresponds with the position of the icon. Thus, the device needs to be told the new screen resolution so it may provide forces at the appropriate region of the device workspace.

[0146] Similarly, a change in aspect ratio can cause distortion in the force feedback sensations. For example, a circle object is displayed on a 4:3 aspect ratio screen and has a force wall surrounding its shape. The circle will feel like a circle on the device if the ballistic frame also has an aspect ratio of 4:3. However, if another monitor is activated so that the screen has an 8:3 aspect ratio, the circle object will feel like an oval due if the screen is directly mapped to the device frame, i.e. if the device frame is " stretched" to fit the screen frame. The use of delta positions 314 prevents this distortion. Since only changes in the mouse position are used to determine cursor position, and not an absolute mouse position in the device workspace, the cursor is positioned without distortion. However, the new aspect ratio still needs to be communicated to the force feedback device since there is a larger area in which the cursor may be positioned. For example, in a 4:3 aspect ratio, the microprocessor might clip a pixel location to the edge of the screen if that location were beyond the edge of the displayed range. However, on an 8:3 aspect ratio, the location should not be clipped since the displayed range is larger.

[0147] To prevent distortion caused by a change in screen size, the ballistic location 336 of the cursor is divided by

a divisor that takes the host screen size into account. For example, if the ballistic frame is 10 times the resolution of the screen frame, then the divisor is 10: the ballistic location 336 would be divided by 10 to achieve a screen pixel position of the cursor if the host screen size had not changed. If the host screen size did change, then the ballistic frame is resized to take the new screen size into account. Preferably, the screen pixel position is equal to the screen size times the ballistic location divided by the ballistic frame size (i.e. the divisor). Thus, if screen resolution on the host has doubled, the divisor would halve, and the screen pixel location would double. The host can also send sensitivity information to the device. If a new sensitivity value is received, the ballistic frame is resized accordingly; as sensitivity is increased, the ballistic frame size is decreased.

[0148] Once the screen pixel location is translated from the ballistic position 336, the screen pixel location is then typically normalized to a standardized range of numbers since communication software and devices have standard values to receive. The normalized screen pixel position 346 is then sent to the host computer as an absolute position in step 350, e.g., as coordinates for display on the screen. The host receives the screen pixel location 346 and displays the cursor appropriately.

[0149] In step 344, the local processor 130 uses the (preferably non-normalized) screen pixel location (and other data) to determine whether any forces should be output, and outputs those forces. For example, the local microprocessor checks the spatial layout of graphical objects which is stored in local memory 134 and determines whether the screen pixel location intersects or contacts any of those objects. If so, there may be a force associated with the interaction, depending on the graphical object intersected. Such a force would be calculated based on a variety of data. Such data may include the device velocity 324, the device position 310, the elapsed time from an event, the distance from a different graphical object in the graphical environment, a history of past mouse motion, etc.

[0150] Once step 346 is complete (and any other desired steps in the provision of forces, which is not detailed herein), the process when needed returns to the measuring of joint angles 302 and joint velocity 320 to begin another iteration of the absolute position reporting process.

[0151] No error correction is required in the present embodiment, unlike the embodiment below, because the force feedback device 11 is operating in absolute mode when the pixel position is determined. The host is no longer receiving simply a change in position (delta) of the user object, but is receiving an absolute coordinate in a coordinate system and need perform no other significant processing (such as ballistics) to display the cursor, i.e., the local microprocessor solely performs the necessary adjustments, such as ballistics, and reports the adjusted position to the host computer. Thus, the host and local microprocessor have the same position value for the cursor, and no opportunity exists for the two positions to diverge.

[0152] FIGURE 11 is a flow diagram illustrating a method 360 of the present invention for implementing enhanced cursor control and indexing in the absolute position reporting mode of method 300. This method provides ballistics for cursor positioning as well as indexing functionality, and many of the steps of method 360 overlap those of method 300. As described above, the mouse workspace preferably includes a central position control region in which the mouse controls the cursor according to a position control scheme. An isometric edge region is provided around the position control region of the mouse workspace. When the mouse is moved into the isometric region, a resistive force is applied to the mouse and the cursor is moved according to a rate control method, where the speed of the cursor is proportional to the amount of penetration into the isometric region. This allows a user to move the cursor throughout the screen frame without causing the user to experience disconcerting interruptions due to the mouse colliding with physical limits to the mouse workspace.

[0153] The method 360 is similar to the indexing methods described above in that a position control region and isometric edge region are provided in the mouse workspace. Method 360 is intended to be used on the preferred embodiment 290 of Figures 9 and 10 and accounts for the use of a ballistic frame on the mouse and for changes in screen size.

[0154] Method 360 begins at 362, in which raw sensor data is read using the sensors 62. In step 364, the joint positions of linkage 40 are calculated from the raw sensor data. In step 366, position kinematics are used to determined the device position (position of the mouse 12 or other user object) in the device frame, as explained above with reference to Figure 10. The last device position is stored in local memory 134 in step 368, and a delta device position is determined in step 370 using the current device position resulting from step 366 and the last device position stored in step 368. From step 366, step 372 is initiated, which checks whether the device position is in the position control region of the device frame. If not, the mouse is in the isometric region and is in rate control mode, where a resistive force is preferably output on the mouse opposing movement into the region. In step 374, the rate control delta is calculated to be the difference between the device position and the rate control edge; this provides the distance of penetration into the isometric region. In step 376, the rate of cursor movement is equal to the maximum rate times the rate control delta divided by the rate control edge width (i.e., the width of the isometric region). The rate control delta divided by the rate control edge width provides the percentage of penetration of the mouse into the isometric region, which is designated to be the same percentage of the maximum rate of the mouse. The maximum rate can be set by the developer; for example, 1000 ballistic pixels per second or 8000 ballistic pixels per second. In step 378, the ballistic position (i.e. the

position of the cursor in the ballistic frame) is set equal to the old ballistic position plus the rate determined in step 376. The process then continues to step 394, detailed below.

[0155] If the device position is in the position control region in step 372, then a position control method is followed. Steps 384-392 implement a preferred ballistics algorithm to provide enhanced cursor control to the user of device 11. This algorithm is generally as follows. If the current device velocity is below the first threshold, the cursor is moved by the smallest amount, e.g., a one-to-one mapping between device movement units and ballistic pixels. If the current device velocity is between the first and second thresholds, the cursor is moved double the amount of pixels as it is moved when mouse velocity is under the first threshold, e.g. a one-to-two mapping between device and ballistic frames. If the user object velocity is above the second threshold, the cursor is moved double the amount of pixels as between the first and second thresholds (a one-to-four mapping between device and ballistic frames). In other embodiments more or less thresholds, or a continuous function, can be used. Also, the host may change the ballistics algorithm in a variety of ways and indicate the changes to the device 11 using ballistic parameters 330 as described for Figure 10.

[0156] Thus, step 384 checks the device velocity so that a ballistic position may be determined. The device velocity is determined in steps 380 and 382, where step 380 determines joint velocities from the sensor data read in step 362 and device velocity is determined from the joint velocities and velocity kinematics in step 382, as described above in Figure 10. In step 384, this device velocity is compared to the ballistic thresholds used in the ballistics algorithm. If the device velocity is within the first ballistic threshold in step 384, then step 386 determines the new ballistic position to be the old ballistic position plus the delta device position (from step 370). The process then continues to step 394, detailed below. If the device velocity is greater than the first ballistic threshold, then in step 388 the device velocity is compared to the second ballistic threshold. If the device velocity is between the first and second thresholds, then in step 390 the new ballistic position is set equal to the old ballistic position plus twice the delta device position. This causes the cursor to move at a faster rate, based on mouse velocity. The process then continues to step 394, detailed below. If the device velocity is greater than the second ballistic threshold in step 388, then in step 392 the new ballistic position is set equal to the old ballistic position plus 4 times the delta device position. The process then continues to step 394.

[0157] In step 394, the screen position of the cursor is set equal to the ballistic position as determined in one of steps 378, 386, 390, and 392 divided by a divisor. The divisor is known from the resolution of the ballistic frame and the resolution of the screen frame, as follows. In step 396, the screen resolution (or size) is obtained from the host. In step 398, the divisor is set equal to the ballistic range (or resolution) divided by the screen resolution as detailed above.

[0158] After the screen position is determined in step 394, it is sent to the host computer in step 399 and used in the display of the cursor in the screen frame. The device 11 also determines and outputs forces, as detailed in Figure 10. It should be noted that the forces on mouse 12 may be determined or calculated differently in rate control (isometric) mode than in position control mode, as described in greater detail in co-pending application serial no. 08/924,462.

[0159] FIGURE 12 is a block diagram illustrating a second embodiment 400 of the present invention for reporting positions to a host computer from a force feedback device. Embodiment 400 provides only relative position reporting from the force feedback device 11 to the host computer 18. This is advantageous in that the force feedback mouse device is detected by the host as a standard non-force-feedback mouse, where the host computer can use standard drivers and other software. This allows the user to control GUI functions even when force feedback is not currently active, such as during startup of the device or during failures of the force feedback functionality. In addition, a relative position reporting process is more appropriate than an absolute position reporting process for such interface devices as trackballs, in which an unrestricted workspace and range of motion is provided for the user object 12. For example, the spherical ball in a trackball controller can be moved in any direction for any number of revolutions and is not as suitable for use with an absolute coordinate system as a device having a workspace with known limits.

[0160] Embodiment 400 provides a device delta position to the host computer from which the host computer determines a cursor screen pixel position. The host computer sends cursor position updates to the force feedback device to provide the cursor position which the device needs to determine when forces are output and to determine the force characteristics. However, since the host cursor position updates occur at a much slower rate, the device must model the cursor position on its own to be able to output forces between host cursor updates. Thus, the host sends ballistic parameters and screen size information so the device can model the cursor position. When the device receives an update, it corrects for any error between its modelled position and the actual cursor position, as detailed below. The host also sends force feedback commands to the device indicate which forces are to be output.

[0161] FIGURE 13 illustrates a process 410 for implementing the second embodiment 400 of Figure 12. A significant difference between method 300 and method 410 of the first embodiment is that the host independently determines a screen cursor position in the present embodiment in parallel with the microprocessor modeling the cursor screen position. Since the host controls the actual position of the cursor, its screen position is considered the "correct" or actual cursor position, while the device modeled value is only approximate. The host must continually send the actual screen position to the device so that errors in the modeled screen position can be corrected and so that the forces can be correctly correlated with actual cursor position.

[0162] Process 410 includes many similar steps to process 300. The current joint angles 302 are measured and normalized in step 304, and the normalized joint angles 306 are processed with kinematics in step 308 to provide a current device position 310. The current device position 310 and previous device position 312 are used to determine a delta position 314, which is reported to the host computer 18 in step 316 over bus 120. As in process 300, this portion of the process 410 functions like a traditional relative mouse, where only a delta value is reported to the host computer. However, unlike the process 300, the delta position 314 is always reported to the host computer, i.e., there is no absolute position to report at any stage of the process. The host computer uses the delta position to position the cursor or other user-controlled graphical object on the display device 20. For example, the host computer adjusts the position of a cursor using the delta amount and its own ballistics algorithm to achieve a new position of the cursor in the screen frame 28. The delta position 314 is sent to the host at a slower rate than the rate of process 410; therefore, in some iterations of process 410, the delta need not be reported to the host. This is because the host only needs to receive a cursor delta position at the rate of displaying the cursor on the screen, which typically is on the order of 60 Hz or every 20 ms. The microprocessor 130, on the other hand, must compute forces at a much higher rate (1000 Hz or above) and thus needs to execute process 300 much faster.

[0163] The current joint velocity 320 also is read in process 410 (both the joint angles 302 and the joint velocity 320 are always measured in the current process). Kinematics are applied in step 322 to obtain the current device velocity 324, and ballistics are applied in step 328 using the device velocity 324, ballistic parameters 330, and delta position 314 to obtain a "modeled" pixel delta 332. The pixel delta 332 is a modeled or predicted value in this embodiment because the host has determined the actual screen position using its own ballistics algorithm and using the delta position 314 provided by the device. The microprocessor 130 preferably uses in step 328 the same ballistics algorithm used by the host computer in its determination of cursor position.

[0164] In step 412, a current pixel location is modeled using the modeled pixel delta 332, a previous pixel location 416, and the host screen size 340, resulting in a current pixel location 414. Step 412 is similar to step 334 of process 300 in that pixel delta 332 and the previous pixel location 416 are used substantially similarly. The host screen size information can be used to determine whether the determined pixel location should be clipped or not. For example, if the aspect ratio has changed from 4:3 to 8:3, double the amount of area is provided. If the pixel location is one that would normally extend beyond the edge of a 4:3 screen area, it would be clipped to the border of the 4:3 screen, but may not be clipped if it is still within the 8:3 area. In one embodiment, if the change in screen size is to a different resolution, the current pixel location can also be adjusted depending on the change in resolution. For example, if screen resolution has been adjusted from 640x480 to 1280x960, then the pixel delta can be doubled before it is added to the previous pixel location.

[0165] Step 412 is different from step 334 in Figure 10 in that a current pixel error 418 is included in the determination of the current pixel location 414. The pixel error 414 is the error between the cursor position modeled by the local microprocessor and the actual cursor position as determined and displayed by the host computer. Although the microprocessor is typically applying the same ballistics algorithm and screen size adjustments to the cursor pixel location as the host computer, errors can be introduced to cause the two cursor location calculations to diverge. For example, in some embodiments, the host computer can "clip" the received delta position 314, i.e., ignore the delta position so that the cursor stays at a constant position. Or, error can be caused by an application program on the host computer moving the cursor based on the program's own criteria and completely independently of the force feedback device. Or, error can be caused by the host computer switching to a different ballistics algorithm, where a delay exists between the host using the new ballistic algorithm and the force feedback device receiving the parameters for the new algorithm.

[0166] The current pixel error 414 is determined by examining the current pixel location 414 at the time of reporting the delta position 314 and a current screen position of the cursor that has been received from the host computer. The host computer periodically reports a current screen position 420 of the cursor to the microprocessor 130, where the screen position is based on the delta position 314 reported to the host in step 316. The reporting of the screen position by the host is triggered by receiving the delta position 314 from the device, and the microprocessor recalculates pixel error when it receives the host screen position 420.

[0167] The microprocessor compares the received screen position 420 with the modeled current pixel location 414 at the time of report 316 (it should be noted that the current pixel location 414 may have a previously-determined error correction value included in its determination, as explained below). If the two locations are the same, no error exists between the two cursor positions, and the microprocessor uses the current pixel location 414 as the location of the cursor in its force calculations and force outputs. If the two locations are not the same, error exists between the two cursor positions and the current pixel location 414 must therefore be corrected. An error correction value is determined as the difference between the current pixel location 414 and the host screen position 420. This value, when added to (or subtracted from) the pixel location 414, will cause the two cursor locations to be the same. Alternatively, if the microprocessor knows the error, then the current pixel location can be adjusted to provide no error.

[0168] The screen position 414 from the host is reported to the microprocessor 130 at a slower rate than the rate of process 410, since the force feedback process must be much faster than the displaying processes of the host computer.

There may therefore be several iterations of process 410 before the screen position 420 is reported by the host to the microprocessor 130. Therefore, once the error correction value is determined after one report of screen position 420, that same value is continually used in iterations providing the determination of current pixel location 414 until the next screen position 420 is received from the host.

[0169] In the preferred embodiment, it is desirable to "smooth out" the correction of the error in pixel location 414 by only incrementally changing the pixel location 414 in each iteration of process 410. For example, if the current pixel location 414 were adjusted with the entire error correction value at one time, this might cause a large change in the pixel location 414. If forces output on the user object are based on the current pixel location 414 of the cursor, then the user may notice the correction of the error by feeling a small jump or discontinuity in the forces applied to mouse 12 after the error is corrected. This is because the forces were determined based on a pixel location having a large error at one point in time, and then the forces are based on the corrected pixel location at the next point in time. Thus, the difference in pixel locations in each iteration is preferably smoothed by only partially correcting the current pixel location 414. With such smoothing, the user does not feel as large a discontinuity in forces when the error in pixel position 414 is corrected. The error correction rate is preferably based on the desired system modeling. The error can be corrected at a rate so that there is no error between the pixel location 414 and the host screen position 420 by the time a new host screen position 420 is received (which may provide a new error, of course). Or, the error can be corrected at a rate that spans multiple received host screen positions 420. Typically, the correction of errors is desired to be as slow as possible to maintain high fidelity of forces to the user, yet the error must be corrected fast enough so as not to create a discontinuity between what the host is displaying on the display device and the forces the user feels on the force feedback device.

[0170] FIGURE 14 is a diagrammatic illustration of an example of the pixel location error correction process explained above. The line 426 represents a time scale, and each vertical mark represents another iteration of process 410. At the first iteration, the pixel location 414 (PL) is determined to be 192 (assuming a single dimension of user object motion) and the error 418 between the pixel location and host screen position is 0. At the next iteration, the pixel location is 200, and the delta position of 4 is reported to the host as in step 316. Over the next iterations, the pixel location is determined to be 200, 201, and 202. At the sixth iteration, the pixel location is determined to be 203; in addition, the host reports a host screen position of 201. Since this screen position was determined based on the delta position 314 reported to the host in the second iteration, the error is measured between the pixel location determined at the second iteration (200) and the host screen position received at the sixth iteration (201). The error value of 1 is then included in the current pixel location, so that the pixel location of 203 is corrected to be 204 based on the error value. This example demonstrates that the pixel location 384 determined by the microprocessor 130, when error exists, is corrected a few iterations behind the actual screen position determined by the host. The microprocessor 130 locally stores the pixel location 414 that was determined in the iteration when the delta position 314 was reported to the host so that the error correction value can be accurately determined.

[0171] It should be noted that, in the example of Figure 14, the error value determined at the sixth iteration continues to be added indefinitely. Thus, if further errors are indicated at later host reports, the new error value must be added to the old error value. FIGURE 15 is a diagrammatic illustration of the error correction value increasing over time. Initially the error correction value is 0, then 1. Over the next host report of the screen position, the current error is 2 pixels, which is added to the previous error correction value of 1 so that the total error correction value is 3. Likewise, in the next host report, the current error is 3 pixels, so that the total error correction value is 6. In one embodiment, if a very large error accumulates, the microprocessor can make the error zero in one iteration regardless of the effect on the user.

[0172] Figure 15 also demonstrates the smoothing or incremental correction of the modeled pixel location in the present invention. An error of 2 pixels exists at the iteration 428. The value of 2 is not added to the previous error value of 1 immediately, since that may cause a force discontinuity to the user. Instead the 2 pixels are added gradually to the error correction value over the next few iterations. For example, the actual value (AV) added to the pixel location 414 is 0 for the first two iterations after iteration 428, then AV = 2 for three more iterations, until the full value of 2 is added in following iterations.

[0173] Referring back to Figure 13, once the current pixel location 414 is determined, the local processor 130 uses that location in step 424 to determine whether any forces should be output and outputs those forces. It should be noted that most forces are determined preferably based on the velocity of the user object in its workspace, not the velocity of the cursor in the graphical environment. Thus, these forces will always feel the same to the user, regardless of any error between the modeled pixel location 414 and the host screen position 420, since they is independent of screen velocity. However, some forces are dependent on interactions in the graphical environment, and will be affected by a cursor position error. Likewise, the ability to initiate or discontinue forces based on interactions in the graphical environment is also affected by cursor position error.

[0174] Once step 424 is complete (and any other desired steps in the provision of forces, which is not detailed herein), the process preferably returns to the measuring of joint angles 302 and joint velocity 320 to begin another iteration of

the process.

Force Feedback Features as Implemented in the Present Invention

[0175] In the above described embodiments, the host computer performs such tasks as the implementation and display of the graphical (or other) environment and cursor, the implementation of multi-tasking application programs, the sending of high-level force feedback commands to the force feedback device, and the reception of position (or other) data from the force feedback device indicating the user's desired interaction with the computer-implemented environment. The force feedback device, in contrast, performs such tasks as reading the position of the user object, receiving high level force feedback commands from the host computer, determining whether a force should be output based on conditions sent from the host computer, and implementing and outputting the required forces when necessary.

[0176] Some tasks, however, may be implemented in either the host computer or the force feedback device, and there are different tradeoffs to choosing one implementation over the other. Events, for example, are one such feature. Another force feedback feature that can be implemented in either the host or the force feedback device is "clipping." Clipping is the selective omission of reported position data when displaying the cursor or other user-controlled graphical object to cause a sensory effect on the user in certain circumstances. Such circumstances include those where displaying the cursor in correlation with the precise user object position would not provide a desired effect. The most common case for clipping is when the cursor encounters a surface or object in the graphical environment which is desired for the user to experience as a hard obstruction or wall. The wall is desired to prohibit further movement into the surface or object; however, due to limitations in hardware, a strong enough force cannot be output on the user object to actually prevent motion into the wall. Thus, clipping is performed: the user object (e.g., mouse) is allowed to be moved by the user into the wall against a resistive force, but the reported position data indicating this movement is clipped or ignored so that the cursor is displayed positioned against the wall surface where it first encountered the wall. The user sees the cursor prevented from moving into the wall and feels a resistive force, and believes that it is a hard obstruction even though the user object can actually be moved into the wall; this is because the user heavily relies on the visual perception of the interaction..

[0177] Clipping can be easily performed by the force feedback device to ease computational burden on the host. For example, the local microprocessor 130 can check whether the user object is being moved into a wall in the graphical environment; if so, the microprocessor can simply discard the position data and report a constant position (or delta) to the host computer that would cause the host to display the cursor against the wall surface. This allows the host computer to be completely ignorant of the clipping process and simply display the cursor at the reported position, thus freeing the host to perform other tasks. However, problems occur in this implementation due to extra burden on the local microprocessor. Since the microprocessor is implementing a very fast force feedback loop in which the user object position is read and forces are output, it is very disruptive to this loop when the microprocessor has to check whether to clip the read position of the user object, and may cause a small distortion in output forces.

[0178] The host computer 18 can also perform clipping in an alternate embodiment. Since it is desirable to keep the application programs 202 and 204, the operating system, and other high level programs ignorant of the clipping process, a lower level program preferably handles the clipping. For example, the translation layer 218 as shown in Figure 4 (or alternatively the context driver or API) can check for the conditions that cause clipping to be applied. If clipping is to be performed, the translation layer alters the input position data appropriately before it is sent on to the context driver, API, operating system and any application programs. Problems with this implementation include increased load on the host with the overhead of intercepting and translating incoming messages.

[0179] A different force feedback feature that can be implemented either by the force feedback device or the host computer is "pressure clicks" or " click surfaces." As described above, these are surfaces, objects or regions in a graphical environment which have additional functionality based on the position of the cursor in relation to the object. Thus, a border of a window can be designated a click surface such that if the user overcomes a resistive force and moves the cursor (or user object) over a threshold distance into the border, a function is activated. The function can be scrolling a document in the window, closing the window, expanding the window size, etc. A similar click surface can be used on an icon or button to select or activate the icon or button.

[0180] The force feedback device can implement click surfaces to ease computational burden on the host computer. The local microprocessor can check whether the cursor has moved into an click surface or region and output the resistive force as appropriate. When the cursor is moved past the threshold distance, the microprocessor reports to the host computer that the action associated with the click surface has been made; for example, the microprocessor reports that a button press or double click has been performed by the user. The host receives a signal that a button has been pressed and acts accordingly. Or, the microprocessor reports that a particular click surface has been specifically selected, where the host can interpret the selection and implement an associated function.

[0181] The host may also implement at least a portion of the functionality of click surfaces. The microprocessor can output the resistive (or other type) force associated with the click surface, but only reports the user object position to

the host. When the host determines that the cursor (or user object) has moved beyond the threshold distance, the host implements the function associated with the click surface (scrolling text, closing a window, etc.) As with clipping, it is preferred that the translation layer handle this functionality by modifying data accordingly before passing it to the context driver and API.

[0182] While this invention has been described in terms of several preferred embodiments, it is contemplated that alterations, permutations and equivalents thereof will become apparent to those skilled in the art upon a reading of the specification and study of the drawings. For example, although examples in a GUI are described, the embodiments herein are also very well suited for other two-dimensional graphical environments and especially three-dimensional graphical environments, where a user would like fine positioning in manipulating 3-D objects and moving in a 3-D space. For example, the isometric limits are quite helpful to move a cursor or controlled object in a 3-D environment further than physical limits of the interface device allow.

[0183] In addition, many different types of forces can be applied to the user object 12 in accordance with different graphical objects or regions appearing on the computer's display screen and which may be mouse-based force sensations or cursor-based force sensations. Also, the various features of the embodiments herein can be combined in various ways to provide additional embodiments of the present invention. In addition, many types of user objects and mechanisms can be provided to transmit the forces to the user, such as a mouse, trackball, joystick, stylus, or other objects. Furthermore, certain terminology has been used for the purposes of descriptive clarity, and not to limit the present invention.

Claims

1. A method for interfacing a multi-tasking graphical environment implemented on a host computer (18) with a force feedback interface device (12, 14) coupled to said host computer, wherein a plurality of application programs are running in said multi-tasking environment, the method comprising:

   creating a context, i.e. a group or set of effects and events, (214) associated with each application program running in said multi-tasking graphical environment, said contexts being allocated in memory of said host computer (18);

   receiving force effect commands from said application programs, said force effect commands commanding said force feedback interface device (12, 14) to output a force effect, i.e. a standardized forces, (222) specified by said command;

   storing said force effect commands into said contexts, wherein each of said contexts is associated with one of said application programs running on said host computer (18), and wherein each of said force effect commands is stored in a context associated with said application program that sent said force effect command; and

   sending said force effect commands in said context of an application program to said force feedback device (12, 14) when said application program is active in said multi-tasking environment.

2. A method as recited in claim 1 wherein said force effect commands in said contexts of inactive application programs are not sent to said force feedback device (12, 14).

3. A method as recited in claim 2 wherein when said application program becomes inactive and a new application program becomes active, new force effect commands are sent to said force feedback device (12, 14) to replace said force effect commands, said new force effect commands being included in a context associated with said new application program.

4. A method as recited in claim 1 further comprising a step of sending force effect commands in a context associated with a background application program to said force feedback interface device (12, 14), wherein said background application program is not active in said multi-tasking environment, such that said force feedback device may output forces based on force effects from said active application program or said background application program.

5. A method as recited in claim 4 wherein only said background application program may send said force effect commands to said force feedback interface device (12, 14) when inactive.

6. A method as recited in claim 1 wherein said contexts further include at least one event that can be triggered by

said force feedback device (12, 14), wherein said event includes conditions based on interactions of a cursor in said graphical environment such that when said conditions are met, an event notification is sent to an application program associated with said event.

7. A method as recited in claim 1 wherein said force effect commands include commands for a spring, a damper, and a vibration.

8. A method as recited in claim 1 wherein said multi-tasking graphical environment includes a Windows® operating system environment.

9. A method as recited in claim 8 wherein said force effect commands are created as effect objects in an application programming interface in said Windows® operating system environment, and wherein said effect objects are sent to said force feedback device (12, 14) when created by said active application program.

Patentansprüche

1. Verfahren zur Verbindung einer auf einem Host-Computer (18) implementierten Multitasking-Grafikumgebung mit einer an den Host-Computer angeschlossenen Kraftrückkopplung-Schnittstellenvorrichtung (12, 14), wobei eine Vielzahl von Anwendungsprogrammen in der Multitasking-Umgebung läuft, und das Verfahren umfasst:

   - Erstellen eines Kontextes, d. h. einer Gruppe oder Menge von mit jedem in der Multitasking-Grafikumgebung laufenden Anwendungsprogramm verbundenen Effekten und Ereignissen (214), wobei die Kontexte im Speicher des Host-Computers (18) zugeordnet werden;
   - Empfangen von Krafteffekt-Steuerbefehlen von den Anwendungsprogrammen, wobei die Krafteffekt-Steuerbefehle die Kraftrückkopplung-Schnittstellenvorrichtung (12, 14) ansteuern, um einen durch den Steuerbefehl spezifizierten Krafteffekt, d. h. standardisierte Kräfte (222), auszugeben;
   - Einspeichern der Krafteffekt-Steuerbefehle in die Kontexte, wobei jeder der Kontexte mit einem der auf dem Host-Computer (18) laufenden Anwendungsprogramme verbunden wird und wobei jeder der Krafteffekt-Steuerbefehle in einem Kontext gespeichert wird, der mit dem Anwendungsprogramm verbunden ist, das den Krafteffekt-Steuerbefehl gesendet hat; und
   - Senden der Krafteffekt-Steuerbefehle in dem Kontext eines Anwendungsprogramms an die Kraftrückkopplungsvorrichtung (12, 14), wenn das Anwendungsprogramm in der Multitasking-Umgebung aktiv ist.

2. Verfahren nach Anspruch 1, wobei die Krafteffekt-Steuerbefehle in den Kontexten inaktiver Anwendungsprogramme nicht an die Kraftrückkopplungsvorrichtung (12, 14) gesendet werden.

3. Verfahren nach Anspruch 2, wobei wenn das Anwendungsprogramm inaktiv wird und ein neues Anwendungsprogramm aktiv wird, neue Krafteffekt-Steuerbefehle an die Kraftrückkopplungsvorrichtung (12, 14) gesendet werden, um die Krafteffekt-Steuerbefehle zu ersetzen, wobei die neuen Krafteffekt-Steuerbefehle in einen mit dem neuen Anwendungsprogramm verbundenen Kontext einbezogen werden.

4. Verfahren nach Anspruch 1, das ferner einen Schritt Senden von Krafteffekt-Steuerbefehlen in einem mit einem Hintergrund-Anwendungsprogramm verbundenen Kontext an die Kraftrückkopplung-Schnittstellenvorrichtung (12, 14) umfasst, wobei das Hintergrund-Anwendungsprogramm in der Multitasking-Umgebung nicht aktiv ist, so dass die Kraftrückkopplungsvorrichtung auf Krafteffekten von dem aktiven Anwendungsprogramm oder dem Hintergrund-Anwendungsprogramm basierende Kräfte ausgeben kann.

5. Verfahren nach Anspruch 4, wobei nur das Hintergrund-Anwendungsprogramm die Krafteffekt-Steuerbefehle an die Kraftrückkopplung-Schnittstellenvorrichtung (12, 14) senden kann, wenn es inaktiv ist.

6. Verfahren nach Anspruch 1, wobei die Kontexte ferner zumindest ein Ereignis einschließen, das von der Kraftrückkopplungsvorrichtung (12, 14) ausgelöst werden kann, wobei das Ereignis auf Dialogeingriffen eines Positionsanzeigers in der grafischen Umgebung basierende Bedingungen einbezieht, so dass bei Erfüllen der Bedingungen eine Ereignismeldung an ein mit dem Ereignis verbundenes Anwendungsprogramm gesendet wird.

7. Verfahren nach Anspruch 1, wobei die Krafteffekt-Steuerbefehle Steuerbefehle für eine Feder, einen Dämpfer und eine Vibration einschließen.

8. Verfahren nach Anspruch 1, wobei die Multitasking-Grafikumgebung eine Windows®-Betriebssystemumgebung einschließt.

9. Verfahren nach Anspruch 8, wobei die Krafteffekt-Steuerbefehle als Effektobjekte in einer Anwendungsprogrammierung-Schnittstelle in der Windows®-Betriebssystemumgebung erstellt werden und wobei die Effektobjekte bei Erstellung durch das aktive Anwendungsprogramm an die Kraftrückkopplungsvorrichtung (12, 14) gesendet werden.

Revendications

1. Procédé de réalisation d'interface pour un environnement graphique multitâche mis en oeuvre sur un ordinateur principal (18) au moyen d'un dispositif d'interface de renvoi de force (12, 14) couplé audit ordinateur principal, où une pluralité de programmes d'application tournent dans ledit environnement multitâche, le procédé comprenant les opérations suivantes :

   créer un contexte, à savoir un groupe ou ensemble d'effets et d'événements (214), associé à chaque programme d'application tournant dans ledit environnement graphique multitâche, lesdits contextes étant affectés dans la mémoire dudit ordinateur principal (18) ;
   recevoir des instructions d'effet de force de la part desdits programmes d'application, lesdites instructions d'effet de force donnant instruction audit dispositif d'interface (12, 14) de délivrer un effet de force, à savoir des forces normalisées, (222) spécifié par ladite instruction ;
   stocker lesdites instructions d'effet de force dans lesdits contextes, où chacun desdits contextes est associé à l'un desdits programmes d'application qui tournent dans ledit ordinateur principal (18), et où chacune desdites instructions d'effet de force est stockée dans un contexte associé avec ledit programme d'application qui a envoyé ladite instruction d'effet de force ; et
   envoyer lesdites instructions d'effet de force présentes dans ledit contexte d'un programme d'application audit dispositif de renvoi de force (12, 14) lorsque ledit programme d'application est actif dans ledit environnement multitâche.

2. Procédé selon la revendication 1, où lesdites instructions d'effet de force présentes dans lesdits contextes de programmes d'application inactifs ne sont pas envoyées audit dispositif de renvoi de force (12, 14).

3. Procédé selon la revendication 2, où lorsque ledit programme d'application devient inactif et qu'un nouveau programme d'application devient actif, de nouvelles instructions d'effet de force sont envoyées audit dispositif de renvoi de force (12, 14) afin de remplacer lesdites instructions d'effet de force, les nouvelles instructions d'effet de force étant incluses dans un contexte qui est associé audit nouveau programme d'application.

4. Procédé selon la revendication 1, comprenant en outre l'opération qui consiste à envoyer des instructions d'effet de force se trouvant dans un contexte associé à un programme d'application en arrière-plan audit dispositif d'interface de renvoi de force (12, 14), où ledit programme d'application en arrière-plan n'est pas actif dans ledit environnement multitâche, de sorte que ledit dispositif de renvoi de force peut délivrer des forces qui sont basées sur des effets de force venant dudit programme d'application actif ou dudit programme d'application en arrière-plan.

5. Procédé selon la revendication 4, où seul ledit programme d'application en arrière-plan peut envoyer les instructions d'effet de force audit dispositif d'interface de renvoi de force (12, 14) lorsqu'il est inactif.

6. Procédé selon la revendication 1, où lesdits contextes comprennent en outre au moins un événement qui peut être déclenché par ledit dispositif de renvoi de force (12, 14), où ledit événement comporte des conditions basées sur les interactions d'un curseur dans ledit environnement graphique de sorte que, lorsque lesdites conditions sont satisfaites, une notification d'événement est envoyée à un programme d'application associé audit événement.

7. Procédé selon la revendication 1, où lesdites instructions d'effet de force comprennent des instructions se rapportant à un ressort, un amortisseur et une vibration.

8. Procédé selon la revendication 1, où ledit environnement graphique multitâche comporte un environnement du système d'exploitation Windows®.

9. Procédé selon la revendication 8, où lesdites instructions d'effet de force sont créées sous forme d'objets d'effet dans une interface de programmation d'application se trouvant dans l'environnement du système d'exploitation Windows®, et où lesdits objets d'effet sont envoyés audit dispositif de renvoi de force (12, 14) lors de leur création par ledit programme d'activation actif.

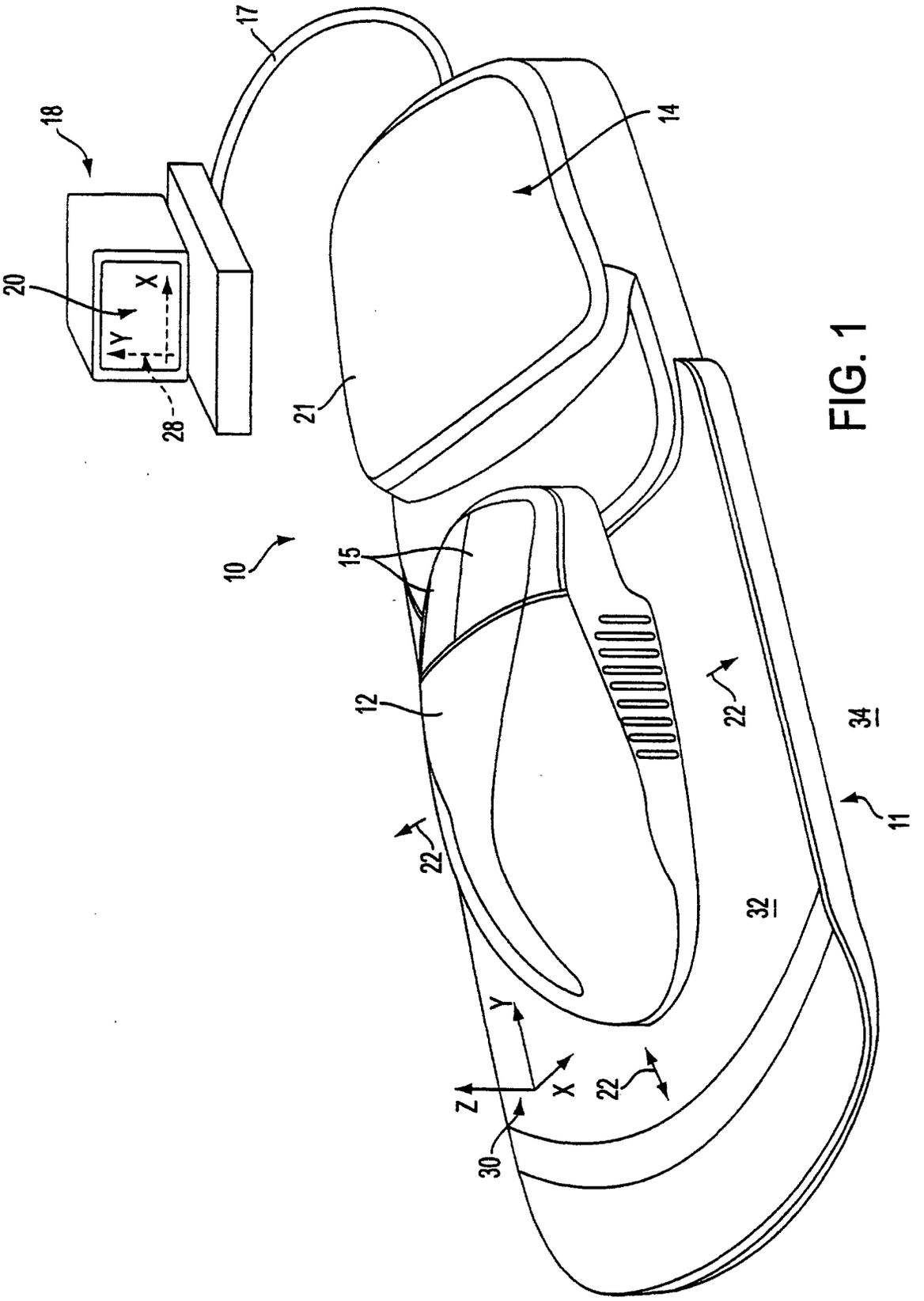

FIG. 1

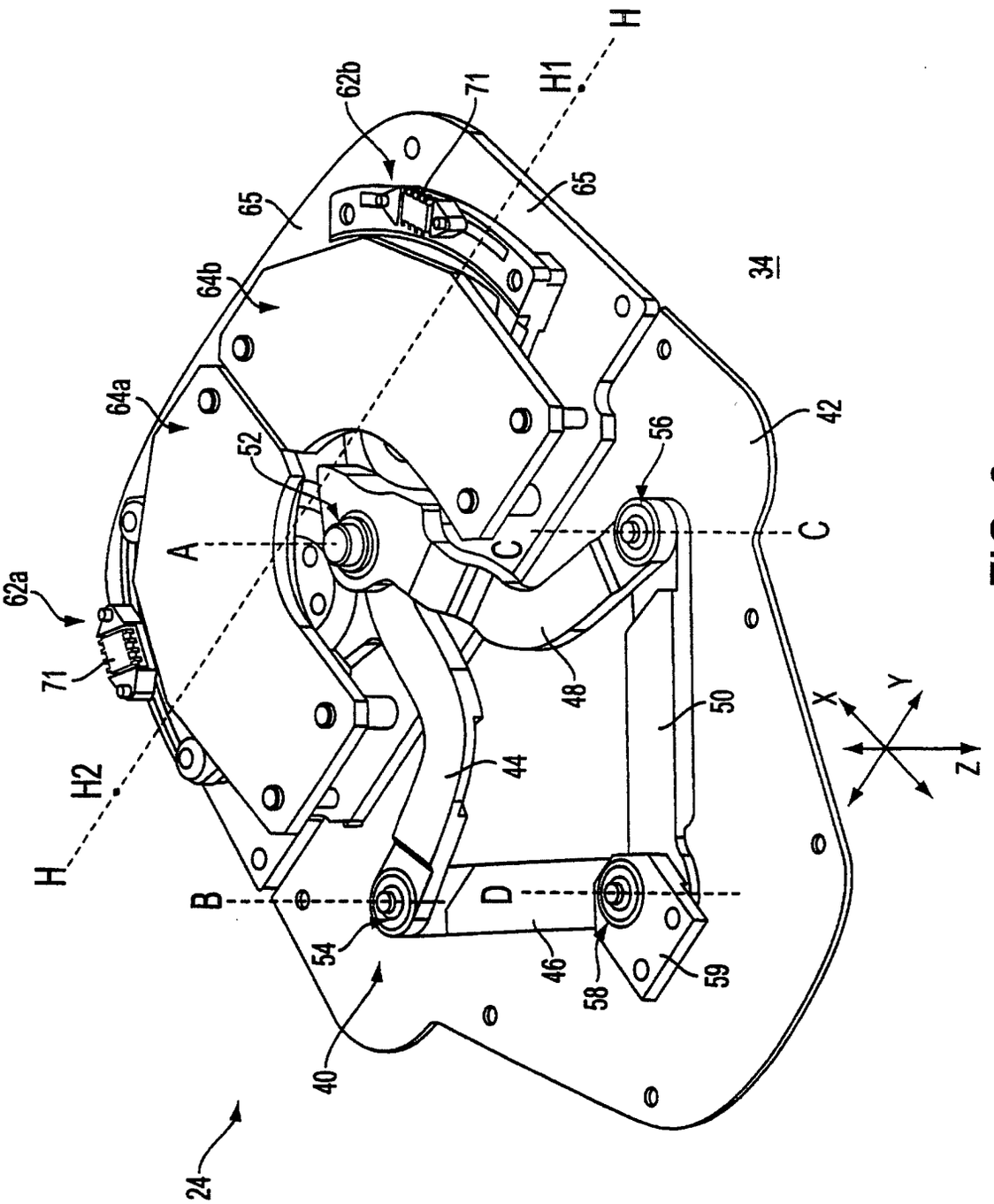
H
H1
62b
71
65
65
64b
34
64a
42
52
56
A
C
C
62a
71
48
50
X
Y
Z
H2
H
44
B
D
54
46
58
59
40
24

FIG. 2

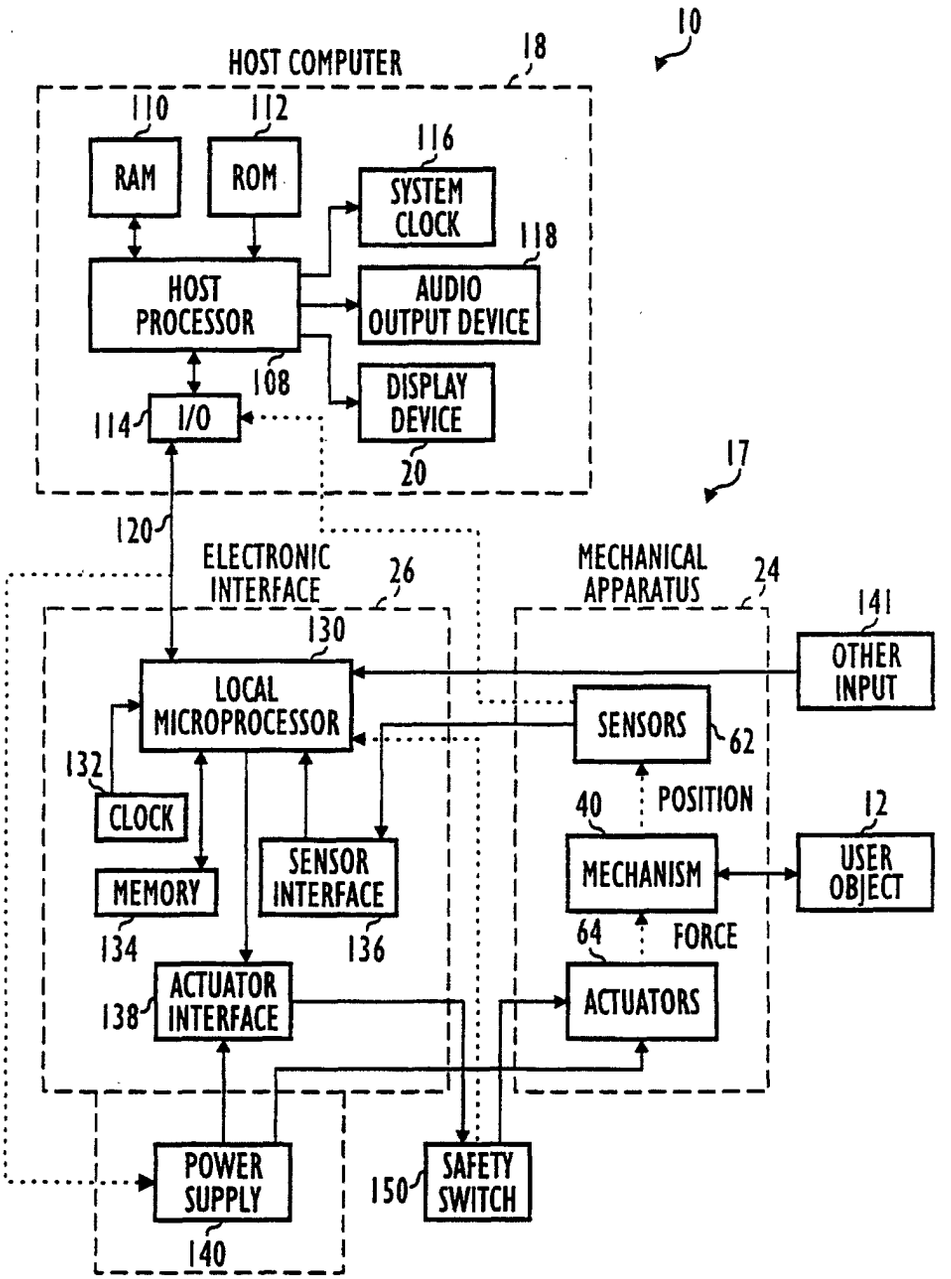
10
HOST COMPUTER
18
110
112
116
RAM
ROM
SYSTEM CLOCK
118
HOST PROCESSOR
AUDIO OUTPUT DEVICE
108
DISPLAY DEVICE
114
I/O
20
17
120
ELECTRONIC INTERFACE
26
MECHANICAL APPARATUS
24
141
130
OTHER INPUT
LOCAL MICROPROCESSOR
SENSORS
62
132
CLOCK
40
POSITION
12
MEMORY
SENSOR INTERFACE
MECHANISM
USER OBJECT
134
136
64
FORCE
ACTUATOR INTERFACE
138
ACTUATORS
POWER SUPPLY
150
SAFETY SWITCH
140

FIG. 3

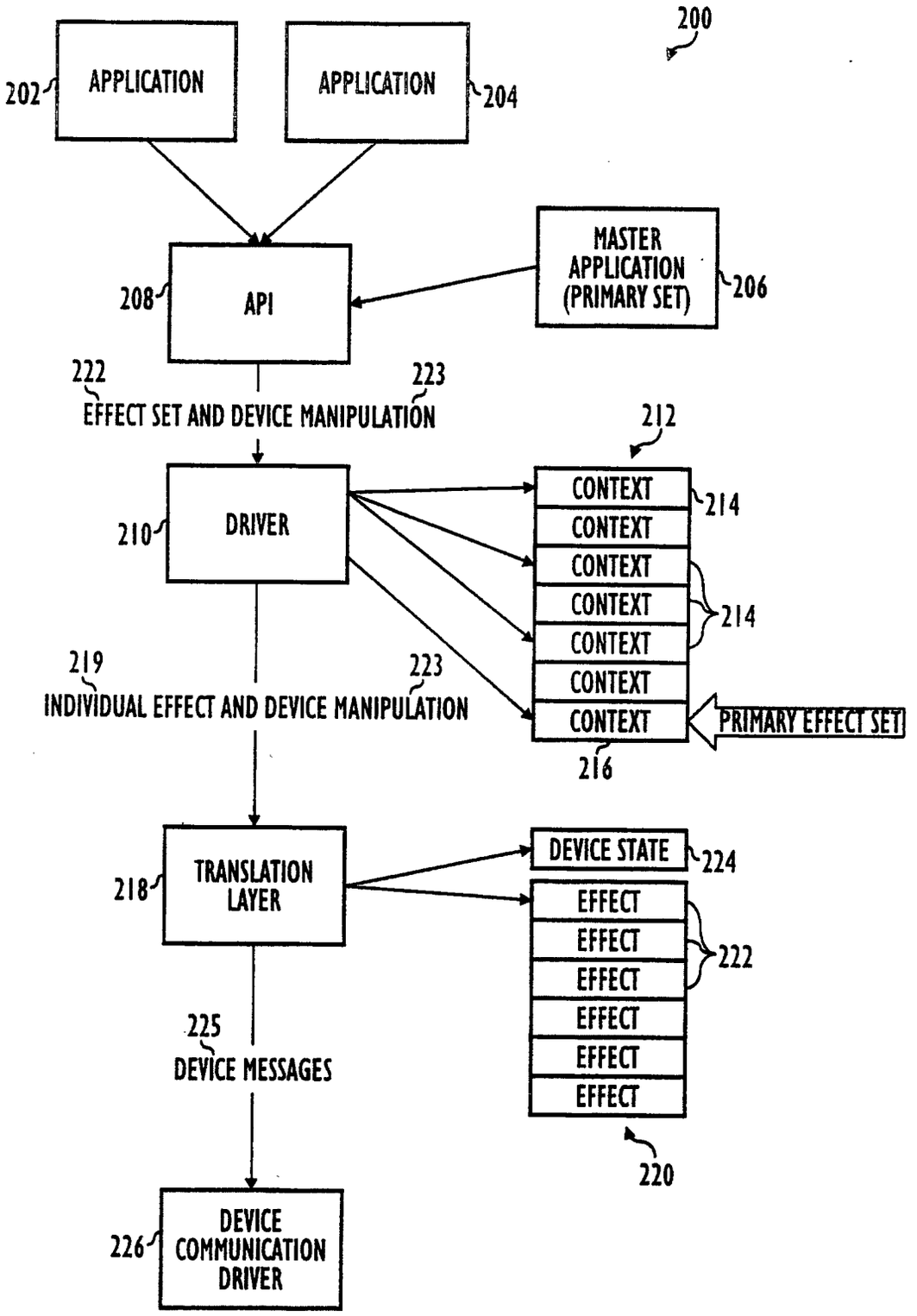
200
APPLICATION
202
APPLICATION
204
MASTER APPLICATION (PRIMARY SET)
206
API
208
222
223
EFFECT SET AND DEVICE MANIPULATION
212
DRIVER
210
CONTEXT
214
CONTEXT
CONTEXT
CONTEXT
CONTEXT
CONTEXT
CONTEXT
PRIMARY EFFECT SET
216
219
223
INDIVIDUAL EFFECT AND DEVICE MANIPULATION
TRANSLATION LAYER
218
DEVICE STATE
224
EFFECT
EFFECT
EFFECT
222
EFFECT
EFFECT
EFFECT
220
225
DEVICE MESSAGES
DEVICE COMMUNICATION DRIVER
226

FIG. 4

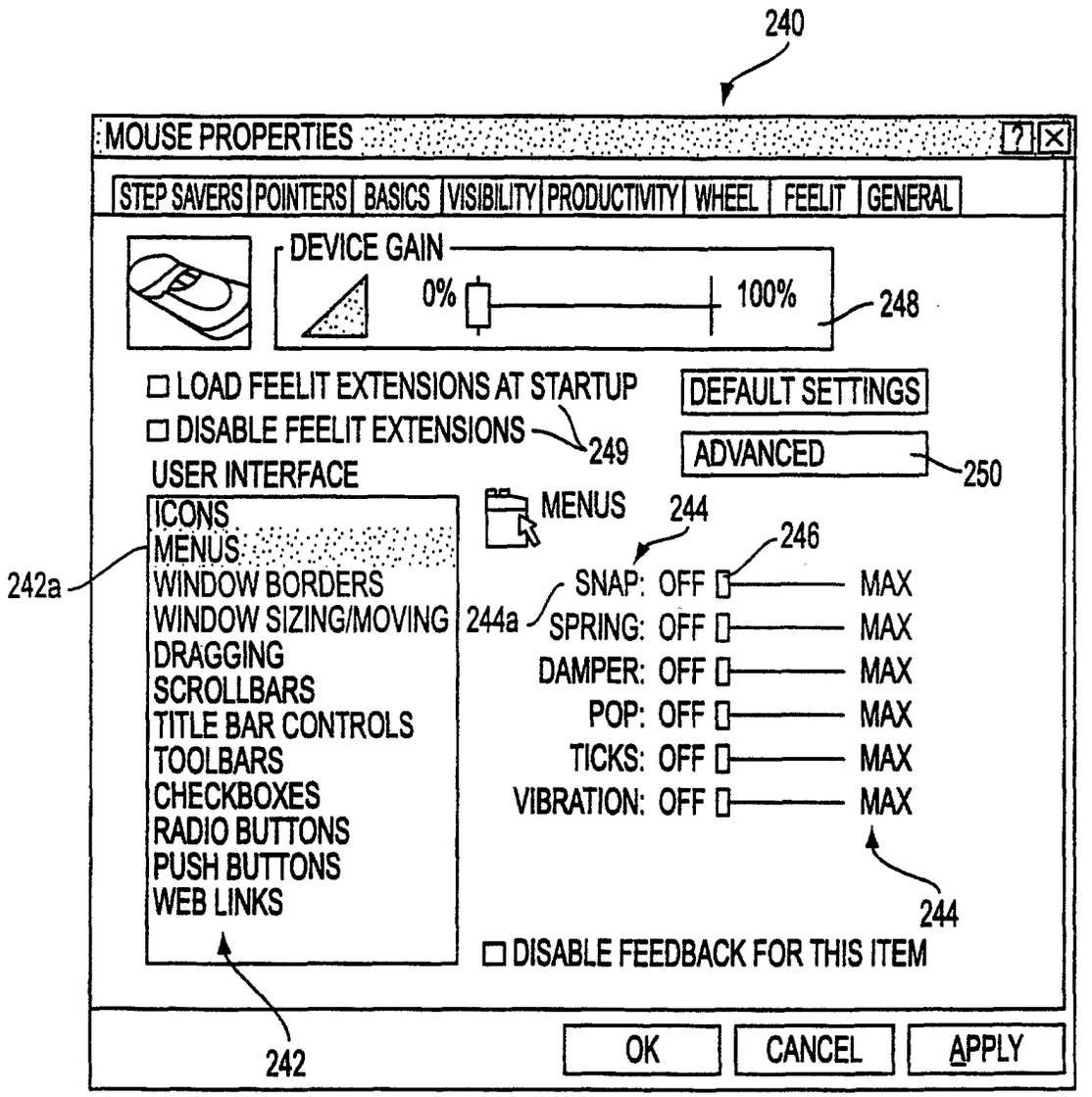
240
MOUSE PROPERTIES
STEP SAVERS
POINTERS
BASICS
VISIBILITY
PRODUCTIVITY
WHEEL
FEELIT
GENERAL
DEVICE GAIN
0%
100%
248
LOAD FEELIT EXTENSIONS AT STARTUP
DISABLE FEELIT EXTENSIONS
249
DEFAULT SETTINGS
ADVANCED
250
USER INTERFACE
ICONS
MENUS
WINDOW BORDERS
WINDOW SIZING/MOVING
DRAGGING
SCROLLBARS
TITLE BAR CONTROLS
TOOLBARS
CHECKBOXES
RADIO BUTTONS
PUSH BUTTONS
WEB LINKS
242a
242
MENUS
244
246
SNAP: OFF MAX
244a
SPRING: OFF MAX
DAMPER: OFF MAX
POP: OFF MAX
TICKS: OFF MAX
VIBRATION: OFF MAX
244
DISABLE FEEDBACK FOR THIS ITEM
OK
CANCEL
APPLY

FIG. 5

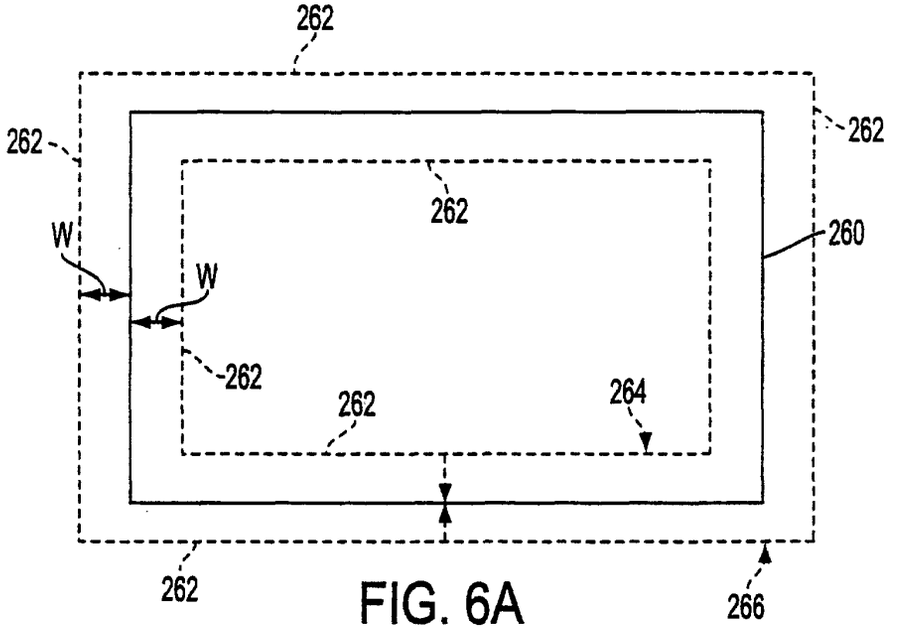

FIG. 6A

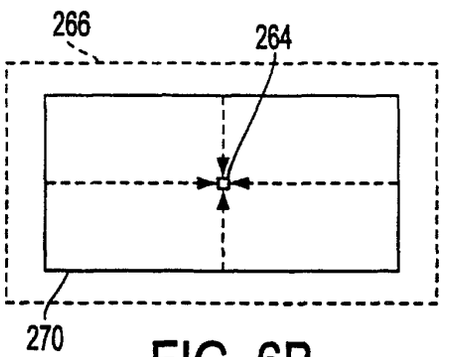

FIG. 6B

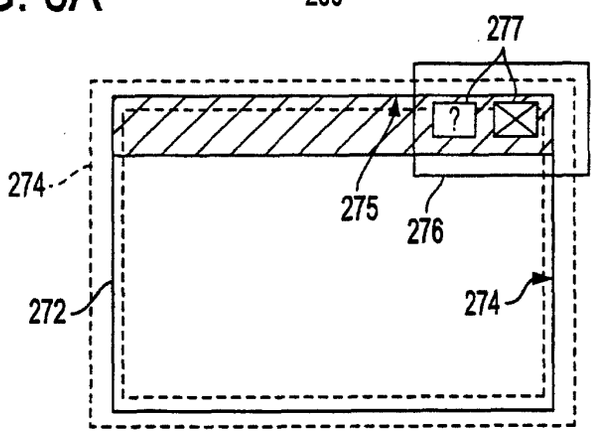

FIG. 6C

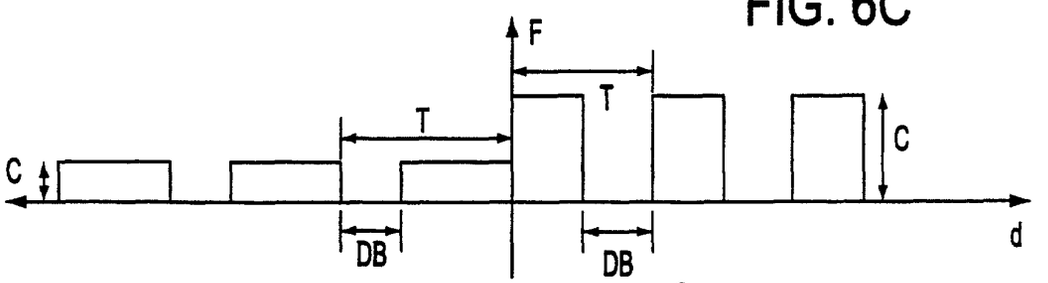

FIG. 7

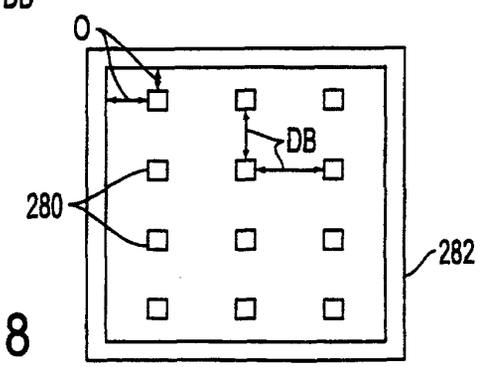

FIG. 8

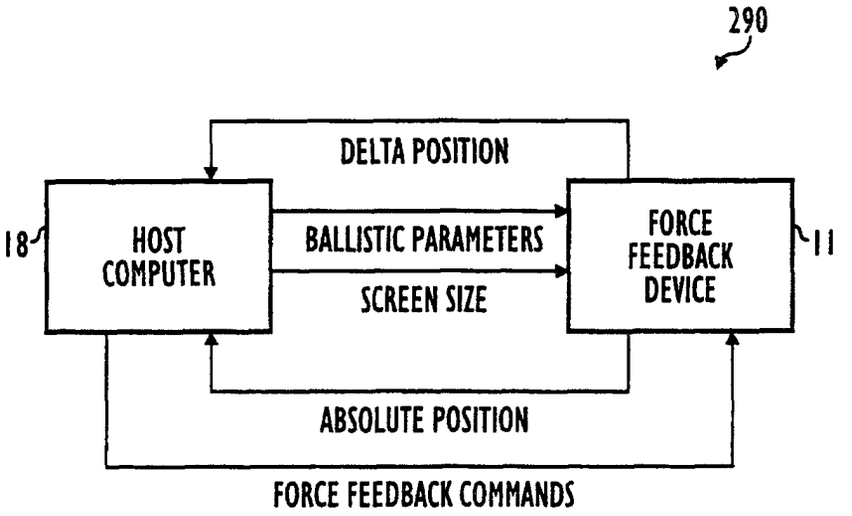
290
DELTA POSITION
18
HOST COMPUTER
BALLISTIC PARAMETERS
SCREEN SIZE
FORCE FEEDBACK DEVICE
11
ABSOLUTE POSITION
FORCE FEEDBACK COMMANDS

FIG. 9

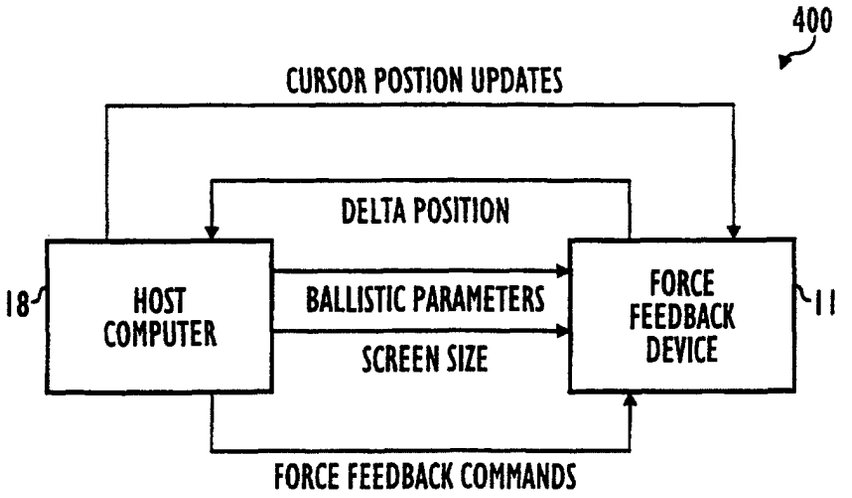
400
CURSOR POSTION UPDATES
DELTA POSITION
18
HOST COMPUTER
BALLISTIC PARAMETERS
SCREEN SIZE
FORCE FEEDBACK DEVICE
11
FORCE FEEDBACK COMMANDS

FIG. 12

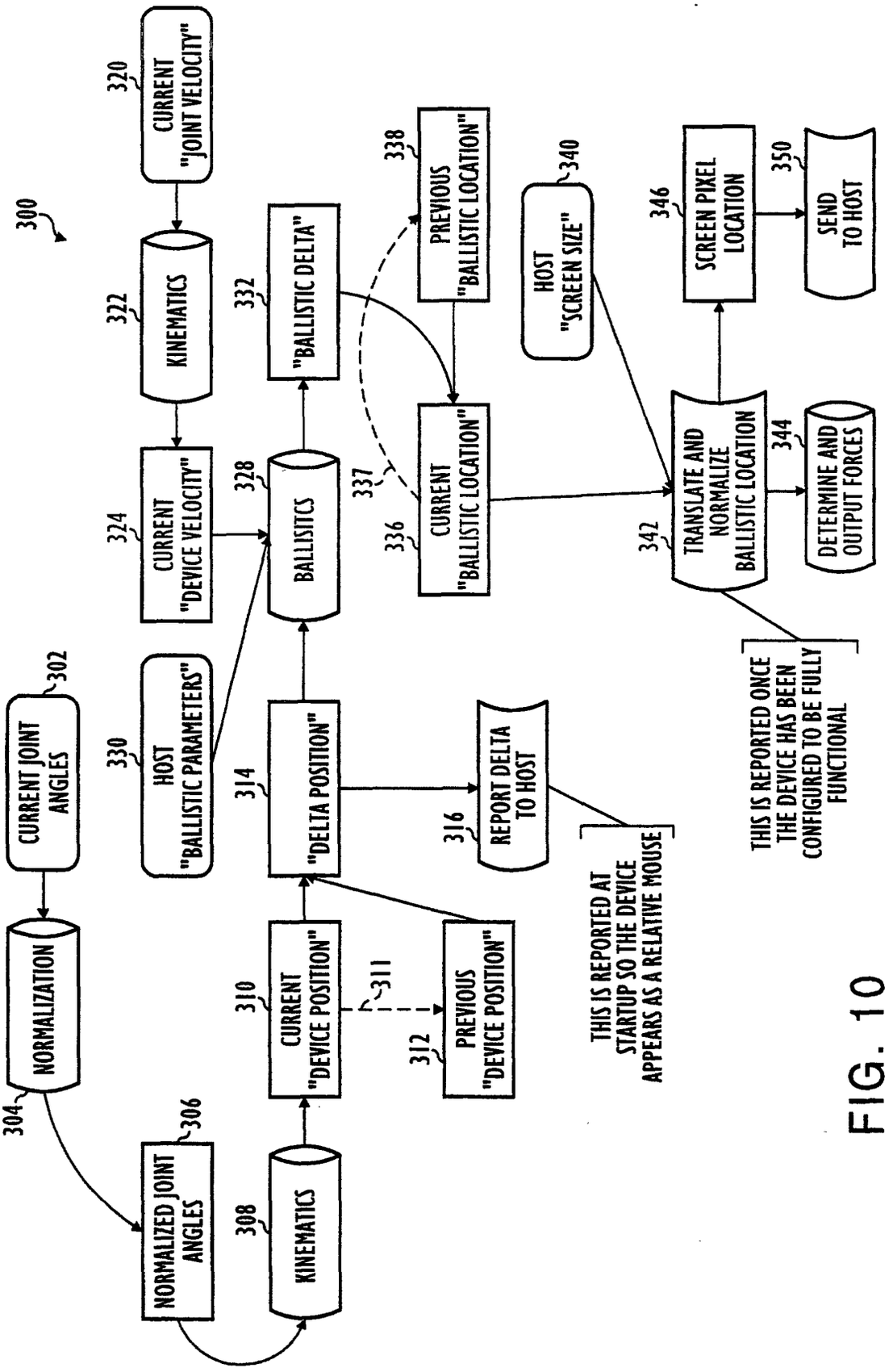
300
CURRENT JOINT ANGLES
302
NORMALIZATION
304
NORMALIZED JOINT ANGLES
306
KINEMATICS
308
CURRENT "DEVICE POSITION"
310
311
PREVIOUS "DEVICE POSITION"
312
HOST "BALLISTIC PARAMETERS"
330
"DELTA POSITION"
314
REPORT DELTA TO HOST
316
THIS IS REPORTED AT STARTUP SO THE DEVICE APPEARS AS A RELATIVE MOUSE
CURRENT "JOINT VELOCITY"
320
KINEMATICS
322
CURRENT "DEVICE VELOCITY"
324
BALLISITCS
328
"BALLISTIC DELTA"
332
PREVIOUS "BALLISTIC LOCATION"
338
337
CURRENT "BALLISTIC LOCATION"
336
HOST "SCREEN SIZE"
340
TRANSLATE AND NORMALIZE BALLISTIC LOCATION
342
THIS IS REPORTED ONCE THE DEVICE HAS BEEN CONFIGURED TO BE FULLY FUNCTIONAL
DETERMINE AND OUTPUT FORCES
344
SCREEN PIXEL LOCATION
346
SEND TO HOST
350

FIG. 10

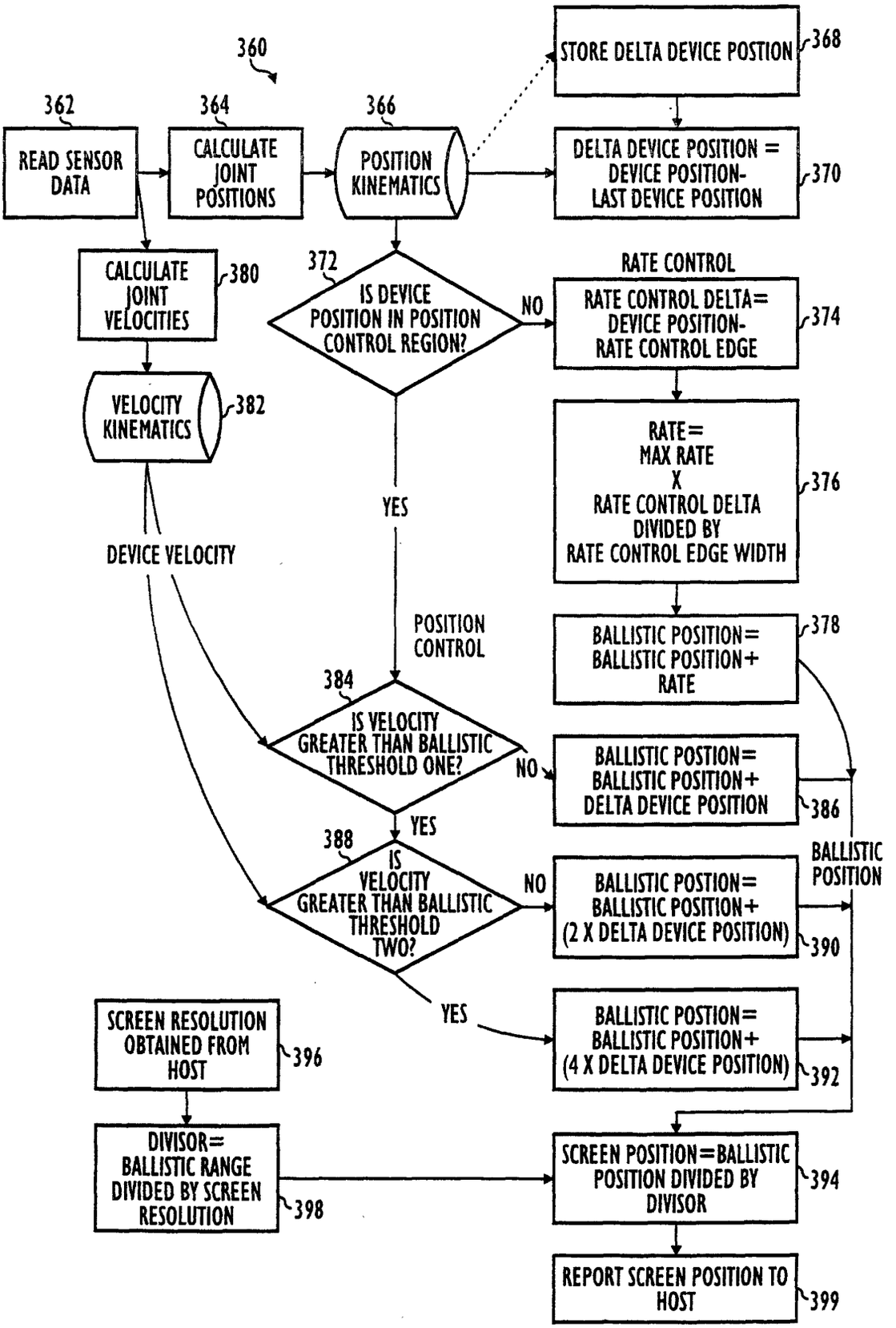
360
362
READ SENSOR DATA
364
CALCULATE JOINT POSITIONS
366
POSITION KINEMATICS
STORE DELTA DEVICE POSTION
368
DELTA DEVICE POSITION = DEVICE POSITION- LAST DEVICE POSITION
370
CALCULATE JOINT VELOCITIES
380
VELOCITY KINEMATICS
382
DEVICE VELOCITY
372
IS DEVICE POSITION IN POSITION CONTROL REGION?
NO
RATE CONTROL
RATE CONTROL DELTA= DEVICE POSITION- RATE CONTROL EDGE
374
RATE= MAX RATE X RATE CONTROL DELTA DIVIDED BY RATE CONTROL EDGE WIDTH
376
YES
POSITION CONTROL
BALLISTIC POSITION= BALLISTIC POSITION+ RATE
378
384
IS VELOCITY GREATER THAN BALLISTIC THRESHOLD ONE?
NO
BALLISTIC POSTION= BALLISTIC POSITION+ DELTA DEVICE POSITION
386
YES
388
IS VELOCITY GREATER THAN BALLISTIC THRESHOLD TWO?
NO
BALLISTIC POSTION= BALLISTIC POSITION+ (2 X DELTA DEVICE POSITION)
390
BALLISTIC POSITION
YES
BALLISTIC POSTION= BALLISTIC POSITION+ (4 X DELTA DEVICE POSITION)
392
SCREEN RESOLUTION OBTAINED FROM HOST
396
DIVISOR= BALLISTIC RANGE DIVIDED BY SCREEN RESOLUTION
398
SCREEN POSITION=BALLISTIC POSITION DIVIDED BY DIVISOR
394
REPORT SCREEN POSITION TO HOST
399

FIG. 11

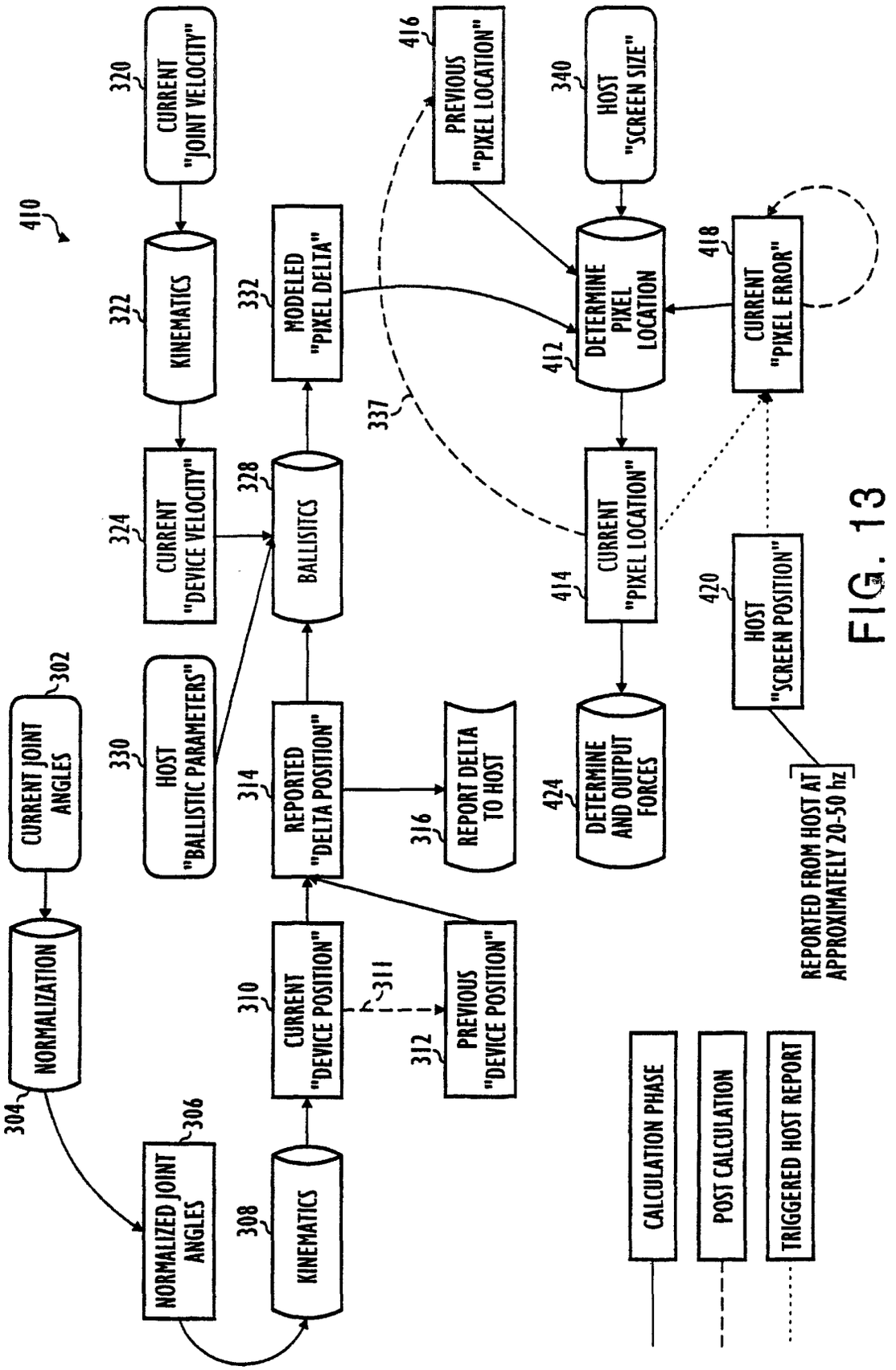
410
CURRENT JOINT ANGLES
302
NORMALIZATION
304
NORMALIZED JOINT ANGLES
306
KINEMATICS
308
CURRENT "DEVICE POSITION"
310
311
PREVIOUS "DEVICE POSITION"
312
REPORTED "DELTA POSITION"
314
REPORT DELTA TO HOST
316
HOST "BALLISTIC PARAMETERS"
330
CURRENT "DEVICE VELOCITY"
324
KINEMATICS
322
CURRENT "JOINT VELOCITY"
320
BALLISITCS
328
MODELED "PIXEL DELTA"
332
337
PREVIOUS "PIXEL LOCATION"
416
DETERMINE PIXEL LOCATION
412
CURRENT "PIXEL LOCATION"
414
DETERMINE AND OUTPUT FORCES
424
HOST "SCREEN SIZE"
340
CURRENT "PIXEL ERROR"
418
HOST "SCREEN POSITION"
420
REPORTED FROM HOST AT APPROXIMATELY 20-50 hz
CALCULATION PHASE
POST CALCULATION
TRIGGERED HOST REPORT

FIG. 13

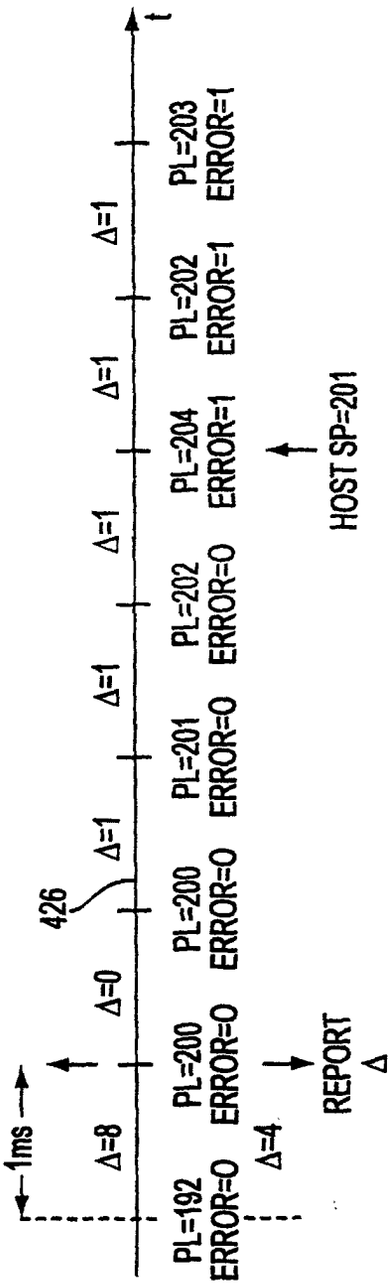
1ms
Δ=8
PL=192
ERROR=0
Δ=4
Δ=0
PL=200
ERROR=0
REPORT
Δ
TO HOST
426
Δ=1
PL=200
ERROR=0
Δ=1
PL=201
ERROR=0
Δ=1
PL=202
ERROR=0
HOST SP=201
Δ=1
PL=204
ERROR=1
Δ=1
PL=202
ERROR=1
Δ=1
PL=203
ERROR=1
t

FIG. 14

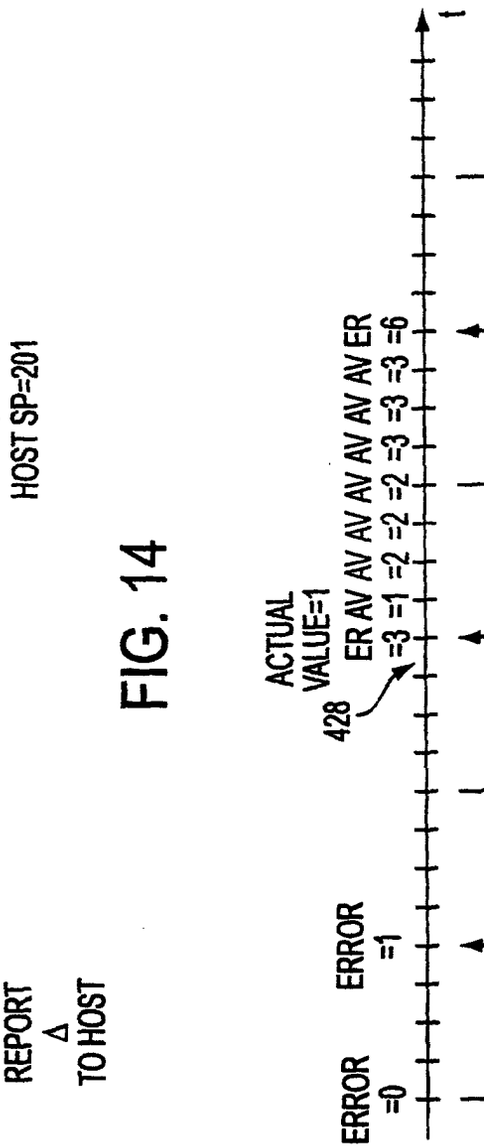
ERROR
=0
ERROR
=1
HOST
REPORTS
SP
REPORT
Δ
HOST
SP
REPORT
Δ
ACTUAL
VALUE=1
428
ER AV AV AV AV AV ER
=3 =1 =2 =2 =2 =3 =3 =6
HOST
SP
REPORT
Δ
REPORT
Δ
t

FIG. 15